CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE STUDY FOR AERIAL VISUAL LOCALIZATION

THESIS

Jedediah Mark Berhold, Captain, USAF

AFIT-ENG-MS-19-M-010

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-19-M-010

CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE STUDY FOR AERIAL VISUAL LOCALIZATION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Jedediah Mark Berhold, B.S.E.E.

Captain, USAF

March 21, 2019

AFIT-ENG-MS-19-M-010

CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE STUDY FOR AERIAL VISUAL LOCALIZATION

THESIS

Jedediah Mark Berhold, B.S.E.E.
Captain, USAF

Committee Membership:

Dr. Robert C. Leishman, PhD
Chair

Dr. John F. Raquet, PhD
Member

Dr. Donald T. Venable, PhD
Member

AFIT-ENG-MS-19-M-010

Abstract

In unmanned aerial navigation the ability to determine the aircrafts's location is essential for safe flight. The Global Positioning System (GPS) is the default modern application used for geospatial location determination. GPS is extremely robust, very accurate, and has essentially solved aerial localization. Unfortunately, the signals from all Global Navigation Satellite Systems (GNSS) to include GPS can be jammed or spoofed. To this response it is essential to develop alternative systems that could be used to supplement navigation systems, in the event of a lost GNSS signal.

Public and governmental satellites have provided large amounts of high-resolution satellite imagery. These could be exploited through machine learning to aid onboard navigation equipment to provide a geospatial location solution. Deep learning and Convolutional Neural Networks (CNNs) have provided significant advances in specific image processing algorithms.

This thesis will discuss the performance of CNN architectures with various hyper-parameters and industry leading model designs to address visual aerial localization. The localization algorithm is trained and tested through satellite imagery of a localized area of 150 square kilometers. Three hyper-parameters of focus are: initializations, optimizers, and finishing layers. The five model architectures are: MobileNet V2, Inception V3, ResNet 50, Xception, and DenseNet 201.

The hyper-parameter analysis demonstrates that specific initializations, optimizations and finishing layers can have significant effects on the training of a CNN architecture for this specific task. The lessons learned from the hyper-parameter analysis were implemented into the CNN comparison study. After all the models were trained for 150 epochs they were evaluated on the test set. The Xception model with pretrained

initialization outperformed all other models with a Root Mean Squared (RMS) error of only 85 meters.

Acknowledgements

To my wonderful wife and children:

Thank you for all the support, love, and encouragement.

Jedediah Mark Berhold

Table of Contents

Page

Page

List of Figures

List of Tables

List of Abbreviations

Abbreviation Page

CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE STUDY FOR AERIAL VISUAL LOCALIZATION

I. Introduction

Aerial visual localization was the first avigation method used in manned flight[1]. Since that time a more accurate and dependable localization tools have been developed, to include the state of the art GNSS, which have furthered the development of unmanned avigation systems. It is possible for adversaries to jam or deny GNSS signals, presenting a renewed need for visual localization in unmanned flight. This thesis evaluates CNN models, that have recently revolutionized image processing in general, as a novel solution to conduct aerial visual localization. Multiple CNN parameters and model architectures will be analyzed on a dataset designed for this task.

This thesis is organized as follows: Chapter I provides a brief overview and objectives this research is attempting to solve. Chapter II goes into the advances of visual avigation, the coordinate systems, and the hyper-parameters and architecture advancements for CNNs. Chapter III discusses the processes used to built the dataset, the system and CNN model architecture, and the methodology to evaluate performance. Chapter IV provides the results of a model hyper-parameter study, and CNN architecture comparison. Finally, ChapterV discusses the conclusions drawn and future improvements to this approach for aerial visual localization.

1.1 Problem Background

Avigation, or aerial navigation, has come a long was since the earliest days of flight where pilots navigated with maps, compass, sextant, and course calculators[1]. We

now have unmanned flight where aircraft navigate without the assistant of pilots using signals from space through GNSS. Unfortunately, GNSS signals can be denied[2]. Visual avigation is part of a solution that could aid the aircraft through a signal disruption environment. In unmanned avigation, if signals cannot be properly sent and received from the aircraft, visual localization must be done algorithmically on-board.

Visual odometry is an effective way to detect changes in position and location. Effective algorithms have been developed by the authors of [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. Visual odometry Artificial Neural Network (ANN) solutions have been developed by the authors of [13, 14, 15, 16, 17, 18] With odometry solutions, errors will exist and slowly perpetuate over time, leading to a lack of global consistency in the position and orientation estimates. The focus of this research is to address these errors though a visual localization CNN.

CNNs are a subset of ANNs which were developed in 1943, but due to the processing complexity they had not been used for mainstream image processing until the authors of [19] outperformed all the other image classification algorithms with a CNN model 'AlexNet' in the 2012 Imagenet competition. Since that time, significant advances have been developed to further improve the performance of CNNs[20, 21, 22, 23, 24, 25, 26, 27, 20, 28, 29, 30, 31].

Some of the improvements in CNNs stem from the extensive work in hyper-parameter development. Advances in methodologies to improve the way weights are initialized in untrained networks to speed up the learning process and improve generalization were developed in [20, 32, 33, 21]. Model optimizers control the process of weight updates during training, and advances in optimizer improvement are observed in [34, 35, 22, 23]. Finishing layers are trade tools to format the CNN output layers into the prediction layer; various finishing layer techiques are shown in [24, 25, 26, 27].

With all this work on hyper-parameter development, which hyper-parameters work best for aerial visual localization? This research will perform a study on the affects of these hyper-parameters for this task.

Since 'AlexNet'[19], the 2012 Imagenet dataset has become the benchmark to test the performance of new CNN architectures. Significant advancements in network size and accuracy have been made in [28, 36, 29, 30, 37, 38, 31]. Can these advancements be leveraged for this aerial visual localization? Which network performs the best for this task? This research will study the training and performance of leading CNN architecture designs for aerial visual localization.

1.2 Research Objectives

This research focuses on the effect of model hyper-parameter and architecture design performance in aerial visual localization. To advance this work, this research attempts to meet the following objectives:

- Establish a reliable dataset to perform side by side model comparisons for visual aerial localization.
- Analyze various CNN hyper-parameters and their effect on the training and testing of a model on the dataset.
- Compare the performance of multiple industry-leading CNN model architectures' in both training and testing on the dataset.

1.3 Limitations and Assumptions

This research is focused on studying the effects of CNN variations on a visual aerial localization dataset. This makes the dataset's affect on this project paramount. The dataset is designed from satellite imagery with nearly sun synchronous orbits[39].

This means that the images are taken around the same time each day. This limits the dataset purely to daytime navigation and excludes more difficult times like dawn and dusk. Sample images had no image enhancement steps, such as contrast adjustment, hue distortion, etc. added. The lack of enhancements could affect the network, by training the model to figure out which satellite took the picture and where the satellite was, as opposed to where the image is in the area of interest. Finally, the satellite coverage over the area of interest is not uniform and there is a higher density of satellite imagery closer to the bounds of the area than in the center.

The dataset limitations were balanced by conducting the training and testing from two separate datasets. This does not remedy the daytime limitation, but a network that trained to figure out the satellite would not be able to translate that skill to the testing dataset. While image enhancement methods could be useful in future iterations, the separate datasets will provide a method to verify the learning of image feature detection. The non-uniform area coverage was accepted as it could tend to pull incorrect network classifications to the extreme values, which could help emphasize the network errors during training.

Additional limitations stem from the CNN itself. While CNN processing has come a long way since the beginning, there remains an extremely high level of processing to train and test the CNN. Consequently, there are major limitations on the input image size. Aerial photography can produce high resolution imagery, and the network operations to process those images would require a massive computing infrastructure. All the models used in this architecture can be run on fairly light-weight systems, but this requires a reduced image size. This research uses a $224 \times 224 \times 3$ image size. Aerial imagery larger than this would require a prepossessing step to format the image to the correct size.

II. Background

This chapter provides information and literature that is relevant to various aspects of this thesis. The sections of this chapter are organized as follows: Section one discusses aspects of aerial navigation or avigation[40] with emphasis on visual navigation and GNSS. Section two discusses coordinate systems with emphasis on the World Geodetic System 1984 (WGS84), Earth Centered Earth Fixed (ECEF), and North East Down (NED). Section 2.3 discusses deep learning with emphasis on ANN, CNN, and advances in CNN design. Sections 2.4 through 2.7 go into further depth on CNN design. Section 2.4 discusses network weight initializations; 2.5 reviews optimizers; finishing methodologies are described in 2.6, and 2.7 discusses specific benchmark CNN designs.

2.1 Avigation

Aerial navigation, also known as avigation[40], is an expansive and diverse field of study. This section reviews topics pertinent to this study namely visual and satellite avigation. Visual avigaton reviews the drawbacks and technological advances in Unmanned Aerial Vehicle (UAV) flight. The GNSS section discusses how satellite systems augment avigation and potential issues.

2.1.1 Visual Avigation

Avigation[40], in the earliest days of flight was essential, but also a challenging problem[1]. Early aviators used basic means, such as a compass, maps, course calculators, and sextants[1] to determine the aircraft's position and direction. These individuals relied on a keen sense of direction and comparing the landmarks they observed from the aircraft's window to those on their maps[1]. This avigation was

eventually supplemented with more advanced instrumentation, such as drift recorders and radio based air position indicators[1], but the ability to identify one's location based on visual avigation remained essential.

Visual avigation is more than simply looking out the window. Aircrew would use all the tools at their disposal to determine their location. They would integrate compass, horizon and slip gyroscopes, airspeed indicators with visually recognized landmarks[1]. Determining ones position from direction and velocity over time from a known point of departure is known as dead reckoning. Aircrew would depend on dead reckoning to get through segments where it was difficult to visually identify a known landmark such as over oceans, farm fields, or with high cloud cover[40]. Visual avigation was rarely used for extensive navigation without augmentation of instrumentation[1].

Avigation was divided into two parts geo-avigation where one would locate objects viewed outside the aircraft, and aerial astronomy which includes navigating by the stars[40]. This thesis addresses the geo-avigation aspect of visual avigation. This methodology has significant challenges such as darkness and obstruction by clouds, similar landmarks, etc. It is difficult for pilots to visually navigate effectively through these conditions without instrumentation augmentation, and even more challenging to design an automated algorithm to visually navigate.

In modern-day unmanned aviation there have been significant advances in automated visual avigation. Aerial visual odometry, using a monocular camera to detect changes in position and location, algorithms were developed in [3, 41, 6, 7, 9, 10, 11, 12]. The authors in [3, 41, 6] provided attitude and position updates through an Extended Kalman Filter (EKF). Automated visual location identification algorithms were developed in [5, 42]. [5] utilized high resolution satellite imagery to develop a Scale Invariant Feature Transform (SIFT) feature database; then with the Inertial

Measurement Unit (IMU) and visual inputs determined a correct location rate of 70%[5]. Visual navigation was used in [43, 16] for formation operations. Robust visual systems have been developed for aircraft landing such as [44, 45] which utilizes the optical flow to determine the distance from the ground. Advances in GNSS denied indoor avigation were illustrated in [46, 41, 6, 7, 47].

There has also been some work in visual odometry utilizing ANNs. The authors in [13, 14, 15] used semi-supervised training develop CNN and Recurrent Neural Network (RNN) visual odometry models using a monocular camera dataset. Authors of [48] utilized multiple CNN models to determine a semantic segmentation of the environment, a global pose regression, and two additional models to determine a visual odometry estimation. Authors of [49] utilized RNNs to generate additional map segmentations utilizing imagery. Work has been done by [50] on location identification based off camera images. Additional work in visual odometry using CNNs and RNNs can be found in [16, 17, 18].

2.1.2 Global Navigation Satellite System

GNSS systems are have become essential to aerial localization[51]. GNSS in avigation is used for communication, air traffic management, aircraft to aircraft operations in addition to localization[51]. GNSS has enabled a reduction in ground-based navigation aids and aircraft avionics[51]. Prior to GNSS Very High Frequency Omnidirectional Range (VOR) stations were placed around the United States for aircraft to triangulate their location. The VOR system is expensive to maintain and is in the process of being decommissioned leaving GNSS to fill in the gaps[52].

Modern GNSS provides much more accurate location information compared to VOR[53]. Unfortunately GNSS and VOR signals can be jammed or spoofed which can deny or provide inaccurate location information[2]. As such an autonomous

military aircraft must be robust enough to manage a GNSS contested environment.

2.2 Coordinate Systems

Describing the aircraft's relation with the surrounding world is essential to relate the aircraft's body frame and the world frame. This thesis will focus on three world frames: the World Geodetic System 1984 (WGS84), the Earth Centered Earth Fixed (ECEF), and the North East Down (NED) coordinate frames. Each one of these coordinate frames has its benefits and drawbacks when relating to the aircraft body frame.

The WGS84 reference frame is a geodetic model used by GPS. WGS84 represents location as the degree offset from the prime meridian, the equatorial plane, and height above sea level, as shown in Figure 2.

Sometimes calculating the world's shape to determine location can be cumbersome. If so, a geocentric coordinate system, such as ECEF, could be a better fit. ECEF utilizes the center point of the Earth as the origin and establishes the x axis along the prime meridian and equatorial plane. The y axis is 90°offset from the x axis also on the equatorial plane, and the z axis is pointing north as seen in Figure 2. The benefit of ECEF is its ability to determine linear distance quickly, and can be useful for satellite and special flight calculations.

When working in a small localized area it may be an adequate to approximate the small segment of the globe as flat, because WGS84, and ECEF make coordinate computations more complex. A localized coordinate system, like NED, could be better in these circumstances. NED establishes a localized plane tangential to Earth at a specific reference point on the surface of the Earth, as seen in Figure 2. Variability from the globe and NED is negligible for a relatively small region and allows calculations to become more intuitive. NED does not work with large globe sections where

relative locations can be distorted as a result of the curvature of the Earth.

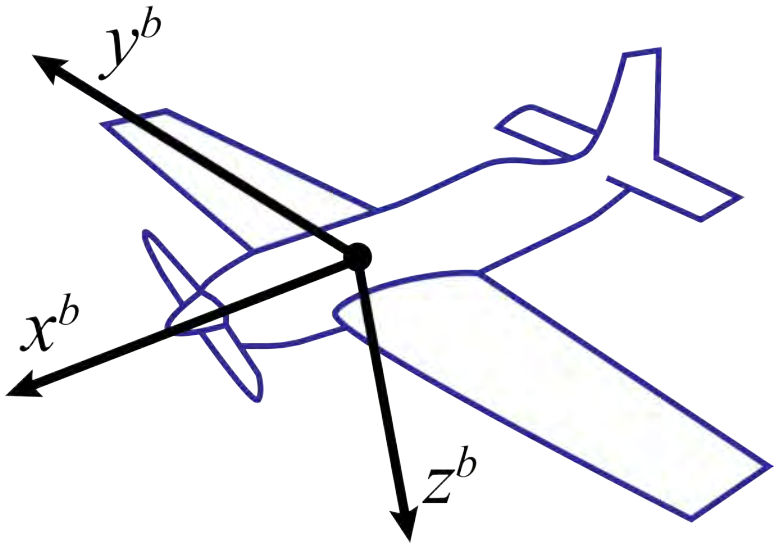

Figure 1. Aircraft body frame.

2.2.1 World Geodetic System 1984

The National Geospatial Intelligence Agency (NGA) determined a geodetic model of the world to be used in United State's GNSS system GPS. A previous geodetic model WGS 72 was insufficient in adequately describing the world's geometry for satellite navigation timing and communication, so the geodetic community came together in the early 1980s to establish WGS84[54]. This update was possible due to extensive altimetry and gravity data from the GRACE satellite mission as well as more accurate geodesy models[54]. The current WGS84 continues to be updated as more precise information is available, and has become the standard reference system due to its accuracy and the global usage of GPS.

The location coordinates in the WGS84 are ellipsoidal. The zero line in the longitudinal direction is the Greenwich meridian, and latitudinal is the Equatorial plane. Longitudinal offsets in Figure 2 are displayed as λ and represent a change in degree on the x, y plane measured as a rotational angle from $-180°$ to $180°$. The latitudinal

offsets are displayed as ϕ, and represent a change in degree in the z direction from $-90°$ to $90°$[55, 56, 57]. WGS84 height variable or h is calculated as the ellipsoidal altitude. A traditional ordering of WGS84 coordinates would be (ϕ, λ, h).

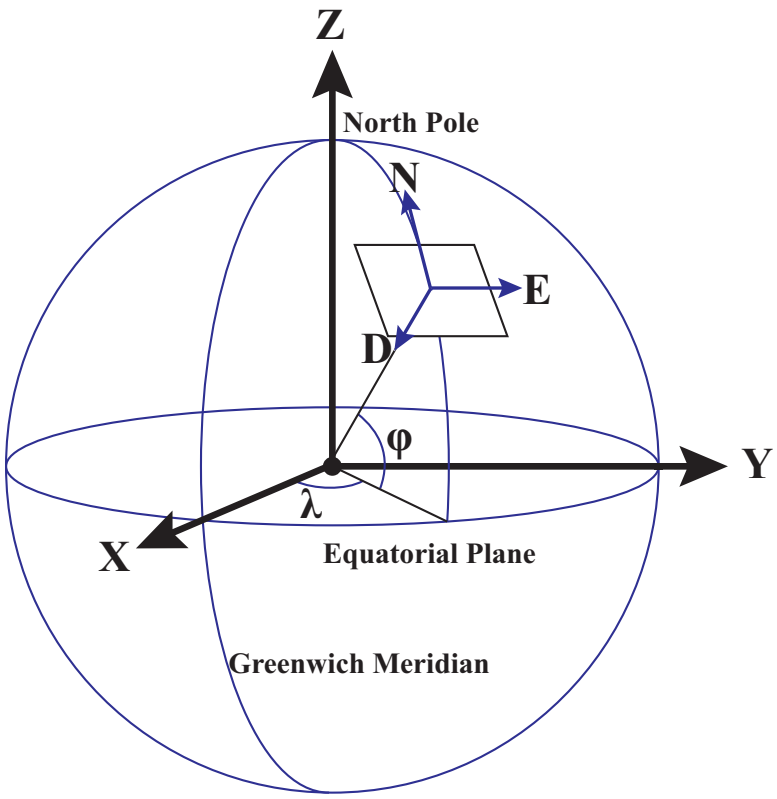

Figure 2. Diagram relating WGS84, ECEF and NED coordinate systems and their relationships. The graphic represents a simplified version of the WGS84 ellipsoid model. Black arrows are ECEF coordinates, and blue arrows are NED coordinate system centered at a specific location on the Earth.

2.2.2 Earth Centered Earth Fixed

The ECEF coordinate system utilizes geocentric rectangular (Cartesian) coordinates (x, y, z) that we learned to love from our mathematics courses[58]. The conversion from geodetic to Cartesian coordinates is seen in Equation 1[59].

$$X = (R_N + h)\cos\phi\cos\lambda$$
$$Y = (R_N + h)\cos\phi\sin\lambda$$
$$Z = \left(\frac{b^2}{a^2}R_N + h\right)\sin\phi \tag{1}$$

In Equation 1, R_N is the prime vertical's radius of curvature and is given in Equation 2. a is the semi-major axes of the ellipsoid, and b is the semi-minor axes of the ellipsoid. ϵ is the eccentricity and it is related to the semi-major and semi-minor axes by Equation 3[59].

$$R_N = \frac{a^2}{\sqrt{a^2\cos^2\phi + b^2\sin^2\phi}} = \frac{a}{\sqrt{1 - \epsilon^2\sin^2\phi}} \tag{2}$$

$$\epsilon^2 = \frac{a^2 - b^2}{a^2} \tag{3}$$

Conversion from ECEF is slightly more difficult and a concise definition is described in [58]. ECEF can be an efficient system when calculating orbits, and can be potentially useful when extensive calculations need to occur with respect to a change in an object's location. ECEF can prove difficult to manage when a localized area is small enough to project it as a flat plane. For this purpose NED would be a better fit.

2.2.3 North East Down

NED is a localized coordinate system used to simplify operations when the working area is sufficiently small that the curvature of Earth is negligible[57]. NED treats the area as a flat plane where the centerpoint of the coordinate system is tangential to the curvature of Earth. This is represented graphically in Figure 2. The x axis points

toward the ellipsoid North, the y axis points to the ellipsoid East, and the z axis points normal to the ellipsoid[56, 57]. The transformation of a point from ECEF to NED is described in Equation 4. $(x, y, z)_{ref,ECEF}$ is the reference point in ECEF coordinates of the origin or center point of the NED coordinate system; $(x, y, z)_{ECEF}$ is the location of the point in ECEF coordinates, and R^{NED}_{ECEF} is the rotation matrix from ECEF from to localized NED frame as seen in Figure 5[56].

$$(x, y, z)_{NED} = R^{NED}_{ECEF}((x, y, z)_{ECEF} - (x, y, z)_{ref,ECEF}) \tag{4}$$

$$R^{NED}_{ECEF} = \begin{bmatrix} -\sin\phi_{ref}\cos\lambda_{ref} & -\sin\phi_{ref}\sin\lambda_{ref} & \cos\phi_{ref} \\ -\sin\lambda_{ref} & \cos\lambda_{ref} & 0 \\ -\cos\phi_{ref}\cos\lambda_{ref} & -\cos\phi_{ref}\sin\lambda_{ref} & -\sin\phi_{ref} \end{bmatrix} \tag{5}$$

Using the NED coordinate system is especially applicable to smaller UAVs as their field of operation is relatively small when compared to the curvature of Earth[56]. When establishing a NED coordinate system it is important to determine the center reference point properly. Often the takeoff position is selected as the reference point for the NED coordinates[56]. Aircraft height h is measured in the $-z$ range for NED.

2.3 Deep Learning

The heirarchy of graphs that builds complex concepts out of layering simple ones is deep learning[60]. Deep learning has recently become popular because of its ability to generalize specific problems better than custom designed algorithms[19]. This section will focus on Artificial Neural Networks (ANNs) and more specifically convolutional neural networks (CNNs) which are the focus of this thesis. It will describe fundamental aspects of modern CNNs that can be used to tailor networks.

2.3.1 Artificial Neural Networks

The concept of a neural network has been around since the early days of computing presented by [61] in 1943. These networks borrowed the biological term 'neurons' to represent weighted activation functions. By connecting a network of these neurons with different weights it was possible to represent specific logic functions. The computational ability of the time was not sufficient for complex tasks, yet incremental advancements were made[62, 63, 64, 65]. The modern viability of the neural network for image processing came with the success of 'AlexNet' in the 2012 ImageNet competition[19]. AlexNet changed image classification standards and created a rush to CNNs as a viable methodology of machine learning.

2.3.2 Convolutional Neural Networks

The CNN was first introduced in 1988[66]. CNNs convolve a weighted kernel matrix across the input, as seen in Figure 3, as opposed to fully connecting all neurons. This practice afforded the network to work well with images and allowed for pattern recognition tasks. Due to the technology of this era computation was difficult for complex networks and CNNs were mostly used for toy problems. A significant advancement came in 1998 with the introduction of gradient descent for network learning by [33]. This provided the basis to update the network and bias weights in a computationally light and effective manner. While CNNs remained computationally heavy at the time, this was a significant advancement in modern CNN training. The attention dedicated to CNNs increased dramatically when [19] outperformed traditional image processing techniques on the ILSVRC-2012 dataset. Since that time many additional developments have occurred to optimize these networks

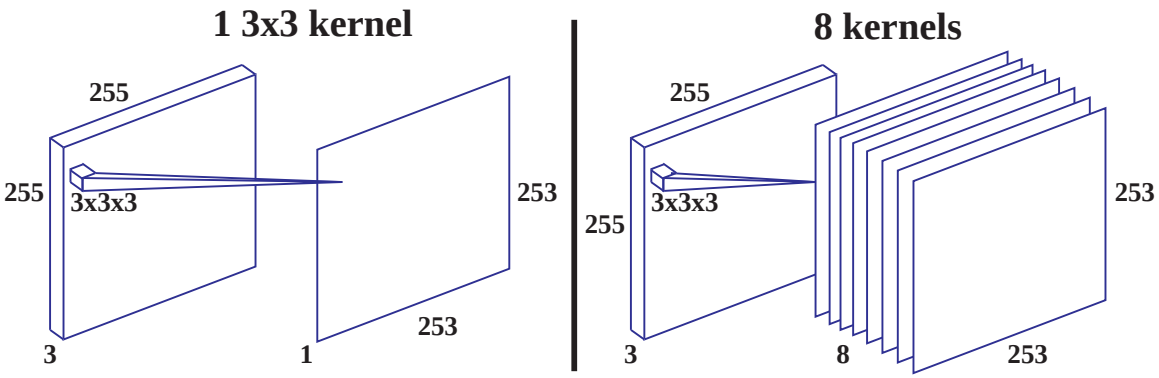

Figure 3. The left side is an example of a $3x3$ convolution with one filter. The extra dimension on the convolution is the input's depth. The right side is a $3x3$ convolution with 8 filter layers. Note that each filter has individual trainable kernel weights.

2.3.2.1 Further Advances in Convolutional Neural Networks

There are an abundance of techniques that have been used to modify and improve CNNs. A comprehensive overview would span volumes, so only specific items that will be a benefit to later topics will be discussed. First will be techniques to preserve dimensionality and techniques to manage reduction and size manipulation. Next will be methods that manage the way weights are updated and normalized during network training. Finally, an overview of how to combine advanced graph structures into usable outputs is reviewed.

A convolution with a $3x3$ kernel size, like that in Figure 3, has an output with a reduced size. This can be useful as the later convolutions require slightly less computations, but can be an obstacle when advanced concatenations are required, such as those in Figure 5. To address this [28, 67, 37, 30] used padding and stride to manipulate the outputs of a convolution layer. 'Same' padding refers to adding zeros at the edges of the input matrix to enforce the same dimensions in the output. Padding allows for advanced directed acyclic graphs without specialized operations to retain shapes. Adjusting the convolution stride affords a quick way to cut the output in half. Strides are the steps taken in a convolution between each kernel. A stride

of one is the traditional convolution that provides the output in Figure 3; a stride of two convolves the kernel with every other member of the matrix. Higher strides are possible, but rarely implemented in practice due to the information loss.

Other network training techniques such as batch normalization and dropout have become a standard in CNNs. Batch normalization as described in [38] is commonplace for CNN architectures [31, 28, 67, 37, 30]. As a network size increases, the effect of weights can saturate the results. Batch normalization is used to reduce this effect in networks. Batch normalization maintains the activation's mean close to zero and the standard deviation approximately one[38]. Dropout is another technique used to reduce over saturation of specific weights[68]. Dropout takes a certain percentage of the output from the previous layer at random and does not pass those weights to the next layer. This forces the network not to depend on a small number of parameters to make major decisions, but spread the decision making across the network[68].

Performing multiple operations from the same input, or combining results from a previous layer, can add great benefits to a CNN[36, 37]. The question is how to combine them back together? There are two popular methods: addition and concatenation. Addition, as used in [37], requires the dimensions of the two layers to be the same and adds the weights of the two in an output layer with the same dimensionality. Concatenation, as used in [36], allows for one of the dimensions to be different from the others and concatenates across the chosen dimension, which, in practice, is typically the dimensionality of the layers. For example, if the weighted output of layer one x_1, with a size of $18 \times 18 \times 4$, and layer two x_2, with a size of $18 \times 18 \times 4$, the residual addition would be $(x_1 + x_2)$ retaining the original dimensions of the input:$18 \times 18 \times 4$. Concatenation would result in x_1, x_2 and expand the dimensionality typically along the last axis; in this case: $18 \times 18 \times 8$

2.4 Initializations

This section discusses methodologies to initialize the various weights for a network. A proper initialization can be taken for granted in CNN infrastructures, but for deep networks they have a significant role to play[32]. A normalized initialization has resulted in reducing the problems of vanishing and exploding gradients[37]. A good initialization can lead to a faster trained network, and some networks need a good initialization to be trained[69]. Advancements in initializers have essentially replaced unsupervised pretraining. A regularizing initializer provides a better baseline for the optimizer and tends to produce improved generalization[32].

Three common initializers are Glorot normal[20], Glorot uniform[20], and orthogonal[20]. Glorot Normal and Glorot Uniform initializers were developed based on best performance through experimentation and monitoring hidden layer weights[32]. Orthogonal initializers developed in [20] determined a scaled random orthogonal initialization reduced the issues of exploding and diminishing gradients while providing significant benefits in the learning process[70].

2.4.1 Glorot Normal

The authors in [20] demonstrated that a carefully scaled random initialization exhibits faster convergence than the traditional arbitrary random initialization. This was the formation of Glorot initializations. The method for scaling standard deviation is displayed in (6) where σ is the standard deviation.

$$\sigma = \sqrt{\frac{2}{inputUnits + OutputUnits}} \tag{6}$$

This initialization provides a truncated random normal distribution, which is centered on zero, and scaled by the input units and output units of the weight ten-

sor. While a pretrained initialization still exhibits faster convergence, the Glorot normal exhibits significant convergence for diverse datasets over a random uniform initialization[20].

2.4.2 Glorot Uniform

Prior to carefully scaled initializers, it was commonplace to perform unsupervised pretraining on neural networks to afford state of the art results[20]. Since the advancement of second order optimizers and better initializer design, unsupervised pretraining is all but obsolete[69]. Currently the default initializer for untrained convolutional kernels in Keras is the Glorot uniform[71]. The Glorot uniform in Equation 7 illustrates the upper and lower bounds to a random distribution which makes up the kernel initialization weights.

$$\pm limit = \sqrt{\frac{6}{inputUnits + OutputUnits}} \tag{7}$$

The number of input units and output units in the weight tensor are utilized to scale the limits of this initializer. Glorot initializers work well for many applications, and it has shown superior performance when ReLU activations are used[69].

2.4.3 Orthogonal

In traditional image processing, filters are designed to extract information from the image. Convolutional filter weights in CNNs perform similar tasks once trained. Establishing an orthogonal initialization has the effect of a pass-through filter at an arbitrary orientation. The orthogonal initialization in [20] is explained in Equation 8 where W is the weight matrix and R is an arbitrary orthogonal matrix, M is a diagonal matrix, and Q are eigenvectors of an input output correlation matrix[20].

$$W = RMQ^T \tag{8}$$

Orthogonal initializations lead to productive gradient propagation in deep linear and nonlinear networks. Under the correct conditions, this initialization provides an amplification of the neural activity through the weights, as well as balancing dampening activity. As the optimizer back-propagates Jacobians, the Jacobians propagate in a nearly isometric manner[20]. These characteristics are especially beneficial in networks dealing with images such as the ones in this thesis.

2.5 Optimizers

One of the great advances in neural networks was the development of improved optimizers. These were a key part in replacing unsupervised pretraining. Second order momentum-based optimizers with carefully scaled initializers have enabled state of the art performance without a pretrained network[20]. These second order optimizers use the process of gradient descent, which is a way to minimize an objective (or loss) function of a models parameters by updating in the opposite direction of the loss function gradient with respect to the parameters[72]. The optimizer's path follows the slope of the loss function surface downhill to a valley[72]. Locating the minimization or maximization requires its parameters to contain a differentiable loss function[22]. Stochastic Gradient Descent (SGD) led to many successes and advancements in deep learning. Because loss functions are composed of a sum of subfunctions evaluated at different data subsamples, SGD takes gradient steps down the individual subfunctions[22]. With noisy data, SGD could have a difficult time locating and often overshoots local minimum[22][72]. SGD does not factor the data characteristics, which led to the development of Adaptive Gradient Algorithm (AdaGrad)[34]. AdaGrad was designed to incorporate the geometry of data previously observed, thus

frequently observed data has a lower learning rate than infrequent data with a higher rate[34]. Unfortunately, AdaGrad produced diminishing learning rates. Three optimizers that address the learning rate issues while capturing the benefits of AdaGrad are RMS prop, AdaDelta[35], and Adam[22].

2.5.1 RMSprop

RMSprop was developed from an unpublished lecture by Geoff Hinton[72]. To address the diminishing gradients from AdaGrad, RMSprop divides the learning rate by a running average of the magnitudes of recent gradients[72]. It uses a discounted history of the squared gradients as a form of preconditioner[73]. RMSprop has become one of the standard methods to train neural networks beyond SGD[74]. It has outperformed other adaptive methods such as AdaGrad, AdaDelta, and SGD in a large number of specific tests[74]. All of these factors have led RMSprop to be a major contributor as a deep learning optimizer

2.5.2 AdaDelta

AdaDelta[35], like RMSProp, utilizes a preconditioner and introduces the additional statistic of the expected squared change of the weights, which rescales the step size proportionally to its history[73]. AdaDelta corrects for the decreasing learning rate featured in AdaGrad by restricting the window of past gradients to a decaying average of past squared gradients. The running average depends only on the previous average and the current gradient[72]. The computational overhead is minimal over SGD[35]. Another advantage to AdaDelta is that an initial learning rate is not an important factor in this optimizer because the dynamic learning rate is computed on a per-dimension basis using first order information[35]. These factors allow AdaDelta to continue adapting the learning rate even after many iterations.

2.5.3 Adam

Adaptive Moment Estimation, or Adam, like AdaGrad, computes adaptive learning rates for each parameter. Also like RMSprop and AdaDelta, Adam saves an exponentially decaying average of past squared gradients[72]. The thing that sets Adam apart is an exponentially decaying average of past gradients, which works similar to momentum[72]. Adam also requires only first-order gradients and has a small memory requirement[22]. Adams advantages over RMSProp are that the magnitudes of parameter updates are invariant to gradient rescaling, which works well with sparse gradients, and performs a form of step annealing[22].

2.6 Finishing

The task for the finishing layer is to convert the shape of the network into a shape compatible for the classification layer. The traditional way to complete this operation is to flatten the outputs of the convolution layers into a single string of values. This flattens the output of the previous layer, yet retains every value. An alternative method, recently becoming popular for classification tasks, is a global average pooling layer; which reduces the dimensionality of each filter into a single value per filter.

2.6.1 Flatten

Using a layer to flatten the outputs of the convolutional layers prior to a dense classification layer, and allows quick management and retention of the convolutional output. This affords the fully connected layer all the information from the previous layer while reshaping to prepare for the CNN model's output. This, depending on the output of the convolution layers, could create cumbersome amount of fully connected weights for the classification layer. The authors in [24] determined that a flattening

layer was less stable during training but it increased convergence speed over a global average pooling layer for their specific task. While average pooling has shown significant advantages in some classification problems, flattening layers show advantages in various applications from adversarial networks to self-driving cars[24][25].

2.6.2 Global Average Pooling

Pooling layers have been commonplace in CNN architecture to reduce dimensionality andextract valuable kernel information. Pooling layers in CNNs summarize the outputs of neighboring groups of neurons in the same kernel map[19]. While pooling layers are commonly used as hidden layers throughout CNNs, a recent trend is to utilize a global average pooling that captures the average of each filter at the end of a deep network prior to a fully connected dense classifier. Global average pooling is utilized in state-of-the-art classification problems. It increased model stability, but hurt convergence speed in [24]. Global average pooling in [26], with the fully connected dense layer improved semantic segmentation results[26]. A global average pooling layer enforces correspondence between feature maps and categories. It also reduces overfitting and is less dependent on dropout regularization[27]. This affords tolerances to vary which can be essential to object recognition[70].

2.6.3 Global Max Pooling

Global max pooling also has its place in CNNs. Max pooling layers are often found throughout CNN architectures as hidden layers, like those found in inception modules[36]. Global max pooling is also evaluated with the same intent as a global average pooling layer. Global average pooling identifies the extent of an object, where global max pooling emphasizes the discriminative parts[75]. While global average pooling outperforms global max pooling for a specific localization task, global max

pooling achieves similar classification performance as global average pooling[75]. Max pooling passes the most dominant features and thus mimics the spatial selective attention mechanism of humans, conferring the more important aspects of an image[70]. Whether average or max, pooling helps to make the representation invariant to small translations of the input[60].

2.7 Benchmark CNN Architectures

This section discusses some of the groundbreaking architectures in CNNs that are relevant to this thesis work. AlexNet propelled the development of modern CNN design[19]. MobileNet focused on smaller applications for deep learning[28]. Inception provides advanced processing with directed acyclic graphs[67]. ResNet introduced adding residuals to reduce diminished gradients, which allows for deeper networks[37]. Inception-ResNet attempted to bring the two technologies together for deeper more connected networks[30]. Xception combined the advances from ResNets and Inception with depth-wise separable convolutions creating a light, yet well performing network[76]. Finally, DenseNet introduces a super connected network to increase the feed-through of earlier layers into the latter[31] .

2.7.1 AlexNet

AlexNet designed by [19] was not used experimentally in this research, but the influences from this early CNN architecture has revolutionized the usage of CNNs. The major advancement from AlexNet was the usage of Graphics Processing Unit (GPU)s for training the neural network. Before this time, training CNNs was extremely time intensive and limited due to the architecture design of the computer's Central Processing Unit (CPU). The CPU is designed to run all the systems operations, and this led to a processor that is the jack of all trades and master of none. The advance-

ment of graphics intensive applications facilitated the need of a specialized GPU that could process the advanced graphics matrix transformations. Utilizing this processing capability is where AlexNet shined. This advancement in training has become the standard that modern CNNs use to train their networks. AlexNet utilized two GTX 580 3GB GPUs [19]. GPUs are well suited for cross-GPU parallelization, and can read and write to the other's memory without the host machine[19]. The authors of [19] took advantage of this in the training which allowed for a larger and deeper network with quicker training times.

Another benchmark advancement from AlexNet was the usage of the ReLU activation function. While there are many non-linear activation functions available, the ReLU has proven to work extremely well with very low overhead in convolutional layers. The implementation of normalization and pooling also aided to a better performance while reducing over fitting. To enhance the dataset and prevent over fitting two forms of data augmentation were performed. The first was generating image translations and reflections of the original dataset, and the second was altering the intensities of the RGB channels in the training images[19]. Enhancing the dataset is important in training to afford the model the ability to 'learn' information that is outside the dataset's shortcomings. Since AlexNet was the predecessor to modern CNNs, it utilized the best optimizer available at the time: SGD. Weights were initialized with a Gaussian distribution and a standdard deviation of 0.01. Finally, AlexNet dramatically outperformed the nearest competitor in the ImageNet competition. The error rate for AlexNet was 10.9% better than the second place method.

2.7.2 MobileNet V2

The MobileNet V2 architecture developed in [28] was designed for mobile resource-constrained systems. This network is an evolution of the previous MobileNet archi-

tecture design[77]. The network was created for computer vision applications, as it decreases operations and memory needed by equivalent performing architectures. MobileNet V2 uses depthwise-separable convolutions which allow similar results as convolutional layers, but decreases the individual layer computations. Instead of having a 3D kernel like the traditional convolution the depthwise-separable convolution convolves each filter independently then uses a pointwise $1x1$ kernel to combine the filters.

One of the great benefits of convolutional neural networks is their effective extraction of non-linearities [28]. In a real multi-dimensional space, $\Re^n$, the ReLU produces a piecewise curve with n-joints. ReLU can work effectively as a linear discriminator in a multi-dimensional space, but when used, information from the channel is lost[28]. This is why MobileNet V2 developed inverted residuals with linear bottlenecks. The linear bottleneck is designed to retain important information in the network and diminish the effects of nonlinearity functions, such as ReLu, from destroying the linear data. The inverted residual pulls the linear bottlenecks to the outside of the depthwise-separable convolutional layers and adds the bottlenecks from a previous segment. Pulling the bottlenecks to the outside has proven to be more memory efficient and increased performance in [28]. This allowed MobileNet V2 to retain the simplicity of MobileNet while significantly improving the accuracy in specific image detection and classification tasks [28]. In comparison to performance and size MobileNetV2 could attain similar performance with MobileNetV1 with using only 200K parameters compared to 800K with MobileNetV1[78, 28] .

The final innovation of MobileNet V2 are the inverted residuals. Residuals are connections from an earlier layer in a network added to a later layer. These connections are effective at battling diminished gradients. As the network back propagates the loss during training, a deep network could have difficulty with the losses being di-

minished and earlier layers receiving minimal, or negligible, updates during training. This methodology first presented in [37] is also used by [28]. A significant difference in MobileNet V2 is the inverted residual. The inverted residual takes the linear bottleneck layer and uses that layer as an expansion layer thus expanding the filters at the begining of each block as seen in Figure 4. The filters of each block can be reduced in each subsequent convolution layer in such a manner as to make the design extremely memory efficient, and also perform well experimentally[28].

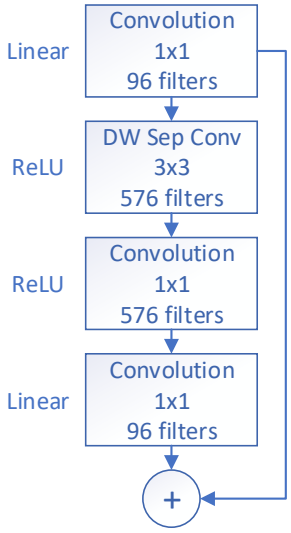

Figure 4. The MobileNet V2 inverted residual with linear bottleneck pulls the bottleneck layer (one that is designed to reduce filters) to the outside of the convolution layers. the linear activation of the bottleneck layers aids the network in retaining linearities because they are residually connected through non-linear layers.

2.7.3 Inception V3

Inception was first presented as GoogLeNet in [36], as an architecture designed to perform even with hardware constraints [67]. The design was first presented in 2014, when such networks as VGGNet, which had three-times the parameters as AlexNet, displayed performance exceeding AlexNet[67]. GoogLeNet, in response, produced similar results as VGGNet with a twelve-times reduction in size from AlexNet[67].

The benefits that Inception provided were through a directed acyclic graph structure. Instead of performing operations linearly and adding additional parameters and complexity, Inception would parallelize the operations and perform convolutions and batch normalizations in parallel then concatenate the outputs.

Inception V3 went a step further and reduced the larger $5x5$ kernel convolutions to two $3x3$ kernel convolutions in series. This saved significant processing resources and still allowed the network to capture some of the advanced dependencies that a $5x5$ convolution would capture. The new modifications also established a methodology to reduce grid size, while expanding the filter banks. This allows for additional complexity while reducing computation time. Further developments reducing convolution computations in the inception layers included those which alternated between $1xn$ layers and $nx1$. [67] selected $n = 7$ for these layers, and in later layers the second $3x3$ convolutions were replaced with parallel $1x3$ and $3x1$ kernel convolution layers, as seen in Figure 5. Inception V3 utilized batch normalization as a regularizer for convolution layers, and had a customized regularization scheme through label smoothing on the classifier level.

2.7.4 ResNet

ResNet, presented in [37], addressed the issues of vanishing gradients. In deep CNNs the weight updates can be disproportionately updated in the latter layers, leaving the early layers untrained. Deeper networks begin to degrade as the depth increase which causes accuracy to saturate[37]. A sweet spot appears, where the depth and training are both optimized, yet this limits network complexity. The solution presented by [37] includes residual connections as shown in Figure 6.

The hypotheses of [37] is that it is easier to optimize the residual mapping than the original non mapped network. This mapping proved to be successful as the perfor-

mance of the network derrived from this theory received first place in the ImageNet competition in 2015. Another benefit to this design is that it can still be trained through standard optimization techniques and implemented with standard CNN libraries without modification. The authors in [37] used Batch Normalization (BN) in between the convolutional layer and the activation, along with the weight initialization techniques described in [21].

2.7.5 InceptionResNet

The excellent performance of ResNet and Inception gave the authors in [30] the idea of putting the two technologies together. In many classification networks the earlier layers focus on shrinking the image filter size, and this network begins with the same intent. As seen in Figure 7, the input width and height is halved in the first convolution, but the depth is increased from three colors to 32 filters. To aid the network in retaining information in size reductions, the InceptionResNet v2 utilizes a methodology of concatenating a max pool and convolution, as seen in Figure 7. This was done to aid the network in retaining information that might be useful in classification by including additional convolution and max pooling filters, yet reduce the filter shape to allow quicker processing.

The InceptionResNet pulls a lot of tricks learned from earlier architecture developers. It includes the Inception and ResNet architecture traits discussed earlier, but it also brings aspects used in mobileNet V2, namely linear activation layers as seen in Figure 8. Figure 9 relates the various blocks explained inf Figures 7 and 8. Inception-ResNet V2 utilizes the same padding on each of the block layers, thus allowing the network to be easily adjustable for different image sizes, which allows concatenation and adding residuals without layer scaling. The total filters increases from Blocks A to Blocks C. Block A begins with 32 to 64 for each of the convolution layers,

and the linear activated convolution layer also works as an expansion layer after the concatenation of the earlier layers with 384 filters. This layer runs five times with the addition of residuals for each layer. Blocks B and C work similarly to A with an increase in filters where B runs 128 to 192 filters with a linear expansion layer of 1154, and Block C has 192 to 256 with a linear expansion of 2048 filters.

The authors in [30] found that deep networks can be trained without residual connections, but residual connections improved the training speed greatly. To reduce network over-fitting a 0.2 dropout was used after the global average pooling layer. To allow the network to be trained on a single GPU, batch-normalization was not used on the summation layers. Removing summations' batch-normalization allowed an increase of inception blocks with the saved processing capabilities. The authors in [30] found that over 1000 filters in residual layers began to develop instabilities in the network, and these results were similar to what was noted in [37]. The authors in [30] developed three networks: InceptionResNet V1, V2, and Inception V4. Inception V4 and InceptionResNet V2 both acheived best-ever performance on the Imagenet classification dataset. Finally an ensemble network (where multiple networks independently run and results are connected) was performed with one Inception V4 and three InceptionResNet V2 which achieved 3.08% top-5 error.

2.7.6 Xception

The basis of the Xception design is inspired by the Inception architecture[76]. The authors of [76] argues that an Inception module performs similar to a traditional convolution and depth-wise separable convolution hybrid. With the success of depth-wise separable convolution in the mobileNet[28] architectures and the relative lightness compared to traditional convolution, the authors of [76] replaced all convolution layers with depth-wise separable convolutions. As seen in Figure 10, the Xception network

appears to more closely resemble the ResNet[37] network than the Inception[67], but because of the depth-wise separable layers the actual functionality is more of a hybrid between the two. Xception is much smaller than the behemoth InceptionResNet[30] and is approximately the size of Inception V3[67] and ResNet50[37]. Benchmark performance on the ImageNet dataset Xception achieved a .945 accuracy compared to .941, and .933 to Inception V3 and ResNet-152 respectively[76]. While the Xception advancement seems only incremental, it does portray the understanding that different linear modules with residuals can operate similar to directed acyclic ones.

2.7.7 DenseNet

What if every convolutional layer in your network had access to the outputs from every previous layer? The authors in [31] decided to do just that. Since the network did not need to relearn redundant feature maps, the authors in [31] argues that the network requires fewer parameters than traditional convolutional networks. A primary difference from DenseNet architecture and ResNet[37], is DenseNet utilizes a concatenation of the previous input with the output of the current layer as opposed to adding the various layers. DenseNet utilizes what the authors in [31] calls a composite function containing batch normalization and ReLU after a 3×3 convolution layer. The output of the composite function is concatenated with the input then passed to the next composite function as seen in Figure 11. The composite function begins with a bottleneck 1×1 convolution layer with 128 filters. This is done to reduce the input feature-maps and make the larger convolution more efficient[31]. This is again followed by a batch normalization and a ReLU activation with the 3×3 convolution layer containing only 32 filters. because of the filter concatenation across the network as seen in Figure 11, each composite function needs to only perform a small piece[31].

The Network performs a specific amount of composite functions in a dense block

then uses a transition layer to compress the network. This compression begins with a 1×1 convolution to reduce the filter dimensionality, then a 2×2 average pool with a stride of two to halve the output size. The performance of DenseNet on the ImageNet dataset was competitive with the other leading networks with a top-5 accuracy of .947 with a multi-crop testing and .939 without[31]. The DenseNet architecture provides an effective way to make each layer in a CNN more efficient and applicable to later convolutional layers and the dense classification layer.

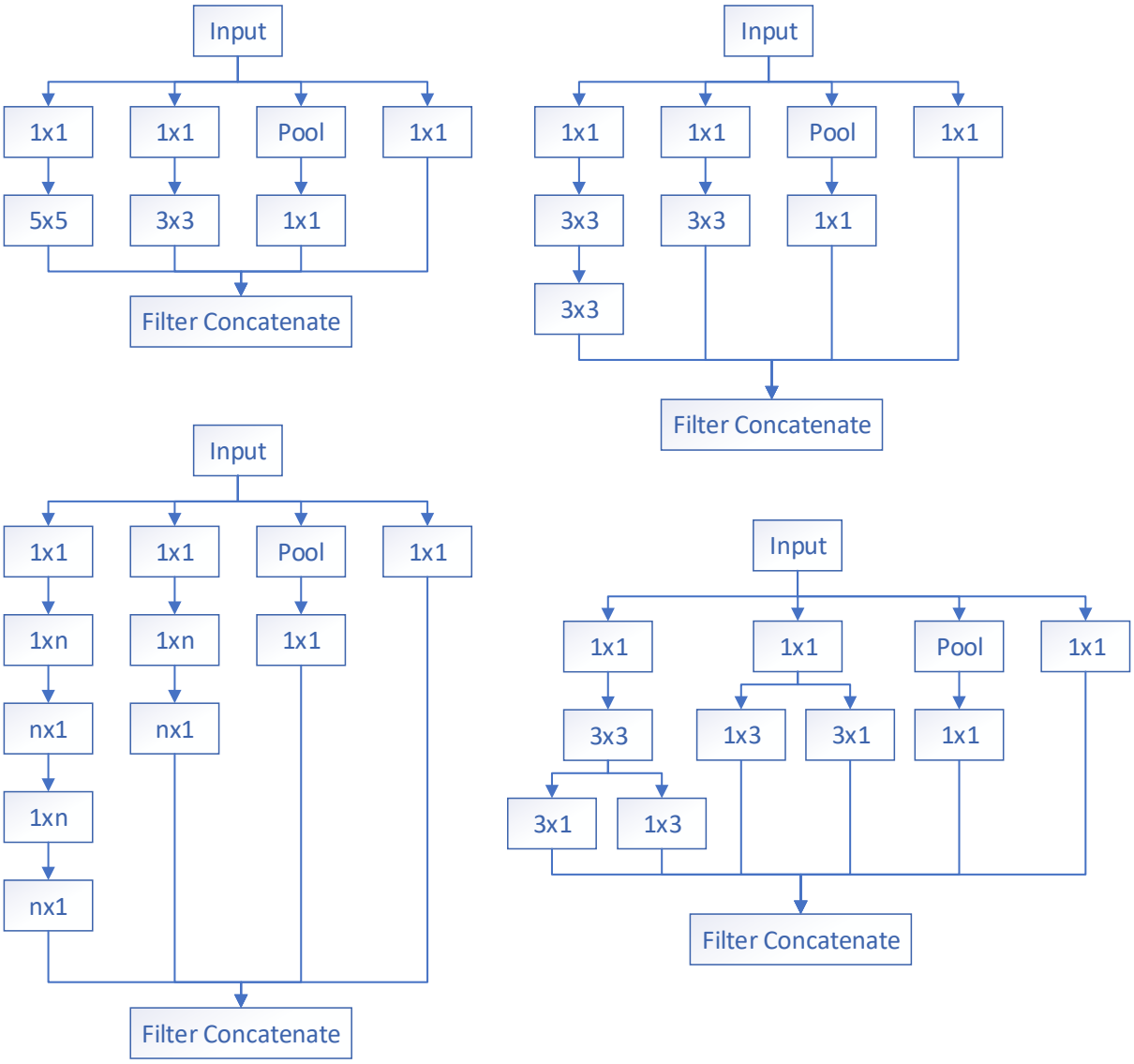

Figure 5. Inception Modules. Top left is the original Inception module. Top right has the $5x5$ convolution replaced with two $3x3$ convolutions. Bottom left is a middle layer where $n = 7$, used to reduce computational complexity. Bottom right is a lower layer used to reduce computations of $3x3$ convolution.

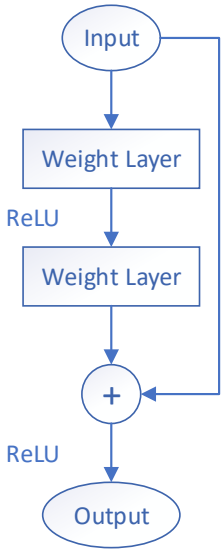

Figure 6. An example of a residual connection. Earlier layers are added to the results of latter layers.

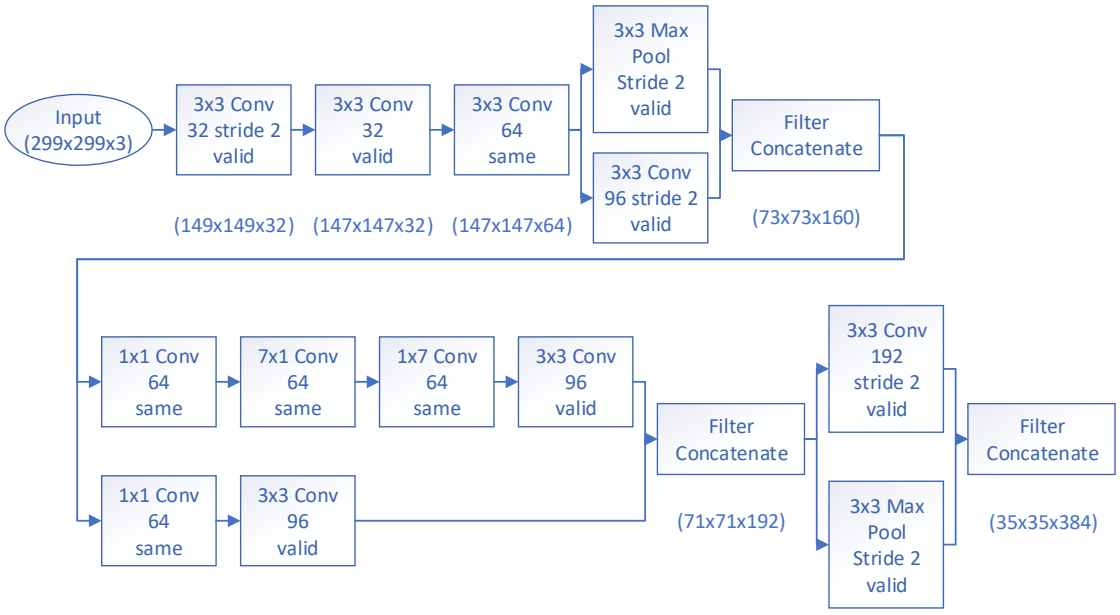

Figure 7. The introduction layers to the Inception ResNet v2. Each convolutional and pooling layer states the kernel size on the top row, the filters and stride on the second, and the padding, either same padding (which retains original shape), or valid padding, which reduces it according to the convolution/pooling output. The text underneath each box is the output size based with relation to the input.

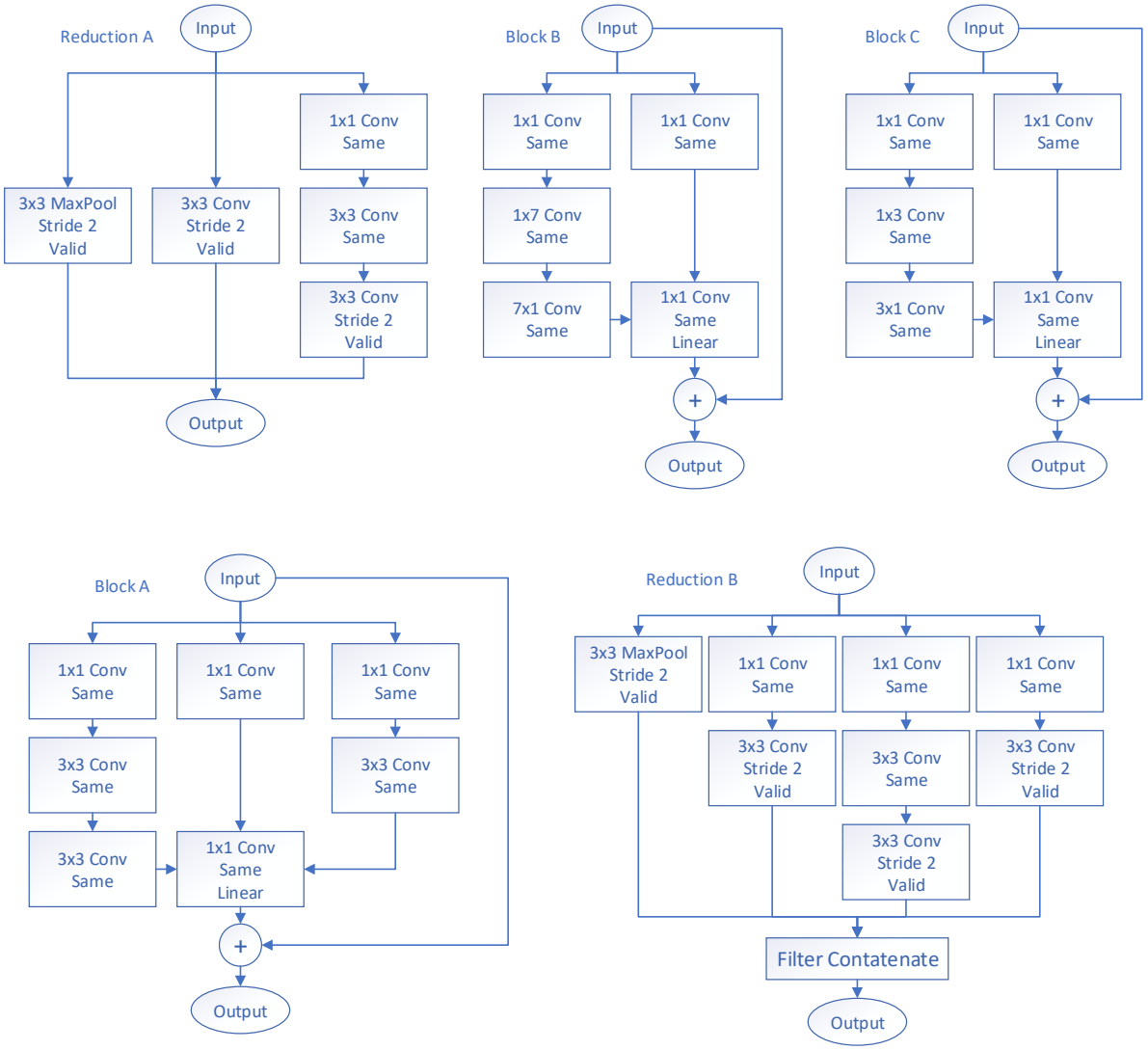

Figure 8. Modules used in InceptionResNet V2. Reduction A and B are used after blocks A and B respectively, and are used to reduce network dimensionality. Each block is tailored for its location in the network.

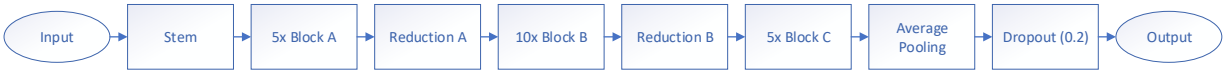

Figure 9. Architecture diagram for Inception-ResNet V2. Blocks based on figure 8 and figure 7

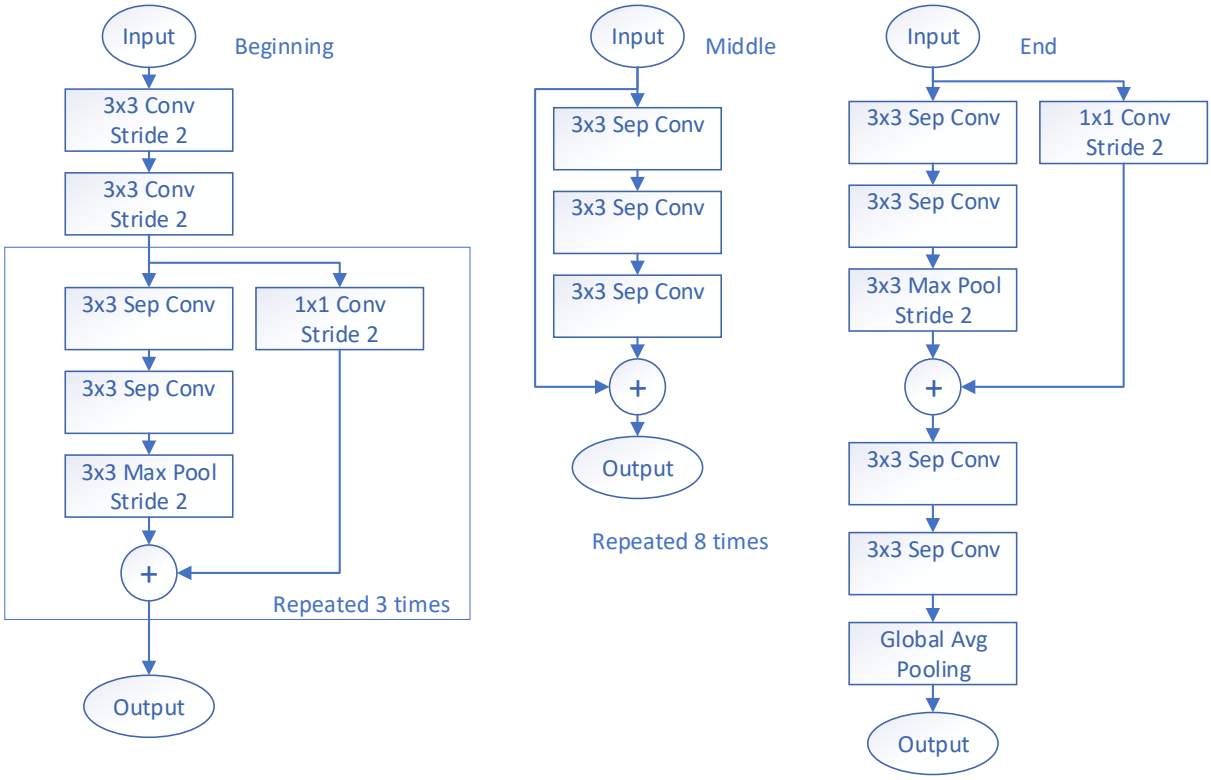

Figure 10. An overview of the Xception architecture. The begining section reduces dimensionality, the middle increases abstraction, and the end prepares the output.

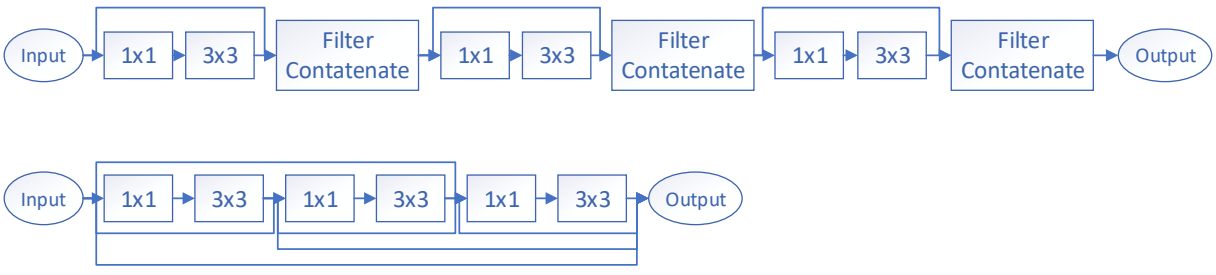

Figure 11. An example of a three composite function dense block from DenseNet. Top is the implementation, and the bottom displays the composite function's connectivity.

III. Methodology

This chapter discusses the techniques and methods used for the experiments in this study. It is composed of five sections: Dataset 3.1, System Architecture 3.2, Hyper-parameter Comparison 3.3, Convolutional Neural Network (CNN) Model Architecture Comparison 3.4, and Custom Loss Development 1.2. The dataset section covers the in-depth origin and formatting of the satellite images to represent an appropriate aerial dataset. Section 3.1 also discusses dataset formatting techniques for the various convolutional neural networks (CNNs). System Architecture details the programming structures, the machine learning architectures, and CNN designs specific to this thesis. The Hyper-parameter Comparison provides the procedure for comparing nine hyper-parameters. CNN Model Architecture Comparison in section 3.4 describes the process for comparing seven innovative CNN models. Finally, Custom Loss Development, section 1.2, discusses a loss specifically designed to integrate the results of a network into an algorithm with Inertial Measurement Unit (IMU) data to provide a more accurate location.

3.1 Dataset

The dataset for this project is built from satellite imagery from multiple seasons and viewing angles. The dataset covers the Dayton, OH area, and is composed of 676 very high resolution satellite images for the training set and 112 for the test set. The images are processed into smaller sizes designed to represent aerial photographs and to be small enough to process adequately in a deep CNN. Each sample image is created using satellite imagery by modeling the view as seen from an aircraft at a specific altitude and orientation. The location coordinates for the center-point of each sample image are localized in a navigation North East Down (NED) coordinate system.

Altitude ranges are based off the image size and area. The following subsections describe the process that was created to take large, raw satellite images and create small sample images that appear similar to how an aircraft would view the scene.

3.1.1 Satellite Images

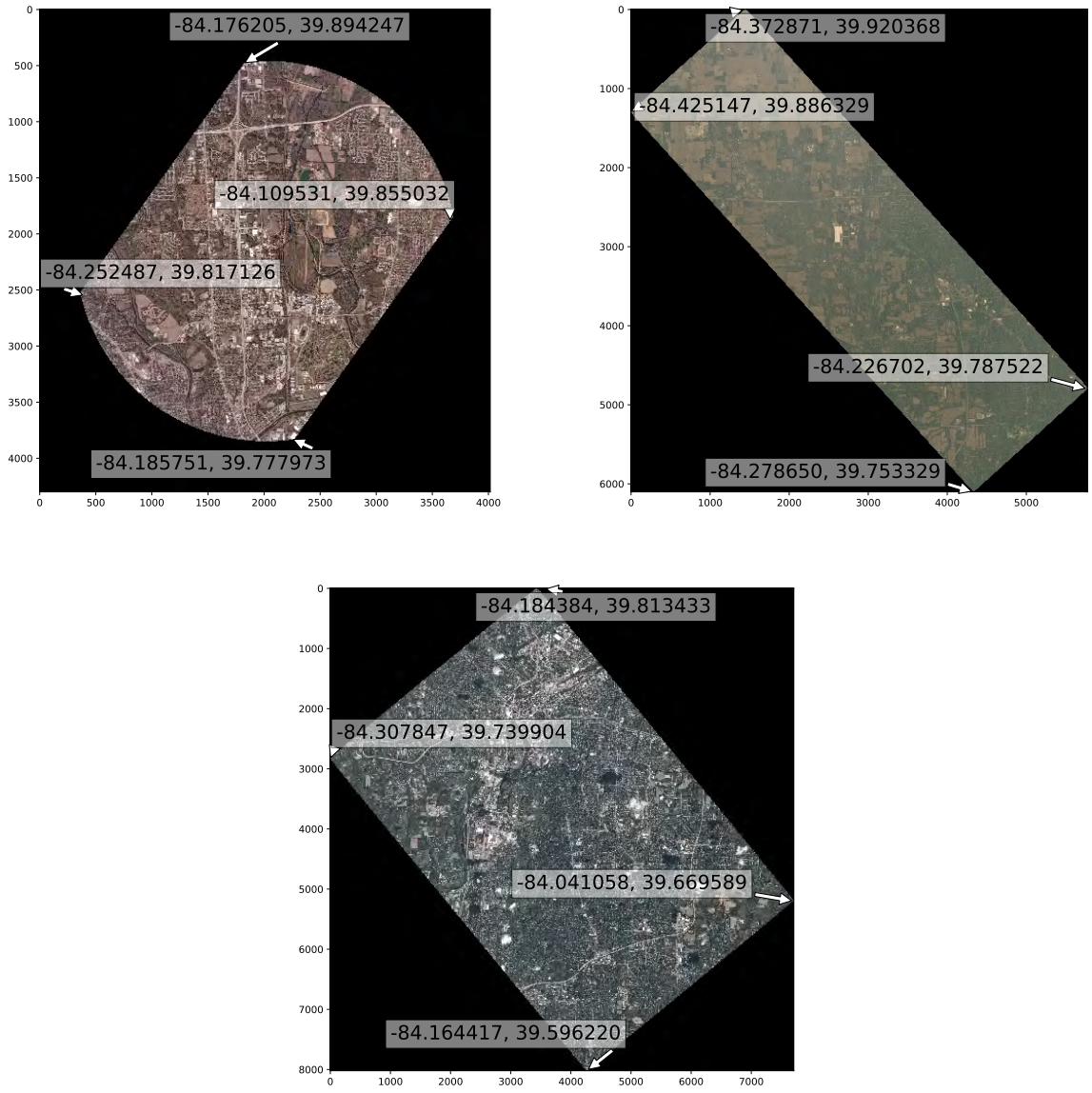

Figure 12. Imagery from the training dataset. Each selected image is from a different satellite. Image 1 is the smallest image in the training dataset and was taken April 2016, image 2 was taken July 2016. Image 3 is the largest image in the dataset and was taken October 2016. The coordinates of the corners are indicated in WGS84.

The dataset used for this project contains spatially-organized raw satellite images that cover 57 total square miles of the area surrounding Dayton, Ohio; 8.08 miles east to west and 7.04 miles north to south. The data was received through a partnership with Air Force Research Laboratory (AFRL), and is sourced from satellite images through AFRL's relationship with Planet Labs Inc. The average raw image size is 139 million pixels, with the largest at 185 million and the smallest at 39 million pixels. The raw images are original footage from various satellites managed by Planet Labs Inc. over the focus area. A sample of the raw images is displayed in Figure 12. Each image has the WGS84 coordinates of its corners stored in a corresponding .json file. Most of Planet Labs Inc's satellites have a low earth, polar or nearly polar, sun synchronous orbit[39] which means that the satellite always has sunlight. On the other hand this causes a major shortcoming, a lack of night images. Since test flights designed to accompany this dataset were intended for day time, the data shortcomings were accepted, but additional work is needed for real world viability. Due to the nearly polar orbits, the satellite images have a rotation with respect to North, as seen in Figure 12.

3.1.2 Location Formatting

The area of interest has a boundary that is a nearly square polygon of Dayton, Ohio. The first step in the processing chain to create sample images is to pass each satellite's boundary coordinates through a geometry based identifier to determine if any portion of the raw image's footprint is within the Dayton bounding box. Only satellite images that contain areas inside the Dayton bounding box are included in the dataset for further processing.

A local navigation coordinate system is established based on the Dayton bounding box boundaries. The center of the bounding box is used for the reference point for the

coordinate system as discussed in Section 2.2.3. Conversion matrices are established to convert the bounding box from WGS84 to Earth Centered Earth Fixed (ECEF) then, finally, to the localized NED, as described in Section 2.2. The satellite image bounding box coordinates are also converted to the localized NED coordinate system.

3.1.3 Image Formatting

The next step is to reformat the raw satellite images that contain sections of the Dayton bounding box into a dataset relating to the problem of visual location identification of an aircraft. The benchmark CNN architectures discussed in Section 2.7 are optimized for image sizes of approximately $224 \times 224 \times 3$ to around $250 \times 250 \times 3$ pixels[28, 67, 37, 30, 76, 31]. In an effort to study the performance of benchmark architectures, selecting a compatible image size is essential. A trimmed image size of $224 \times 224 \times 3$ from these specific satellite images produces an image approximately 700 meters from edge to edge. One popular small Unmanned Aerial Vehicle (UAV) camera contains an angular field of view of 94°[79]. Equation 9 can be used to determine the simulated aircraft's altitude (working distance), given the width of the image and the angular field of view of a lens.

$$WorkingDistance = \frac{HorizontalFOV}{\tan \frac{AngularFOV}{2}} \tag{9}$$

The minimum altitude would be 652 meters or 2100 feet. A camera with an angular field of view of 140° would decrease the minimum altitude to a more usable 254 meters or 833 feet. Camera lens zoom and cropping could be utilized for higher altitudes.

The boundaries explained in Section 3.1.2 were used to determine the satellite photographs that contain the bounding box. Then portions of each raw image that were within the bounding box were extracted to be further subdivided into image

samples. A circular radius was used to subdivide the photograph into smaller segments. Then, a $224 \times 224 \times 3$ cutout was taken at a randomized rotation centered within the circle. The whole process is described in Figure 13. Insead of creating and storing all the raw satellite subsample images, the algorithm saves the directions to cut each sample and returns that list to generate only the samples needed. The list of samples was randomly shuffled and a dataset of 100,000 sampled images was returned for the training dataset with a corresponding list of the centerpoint of each sample in NED coordinates. The images and the centerpoint coordinates were returned as the input and output for the CNNs.

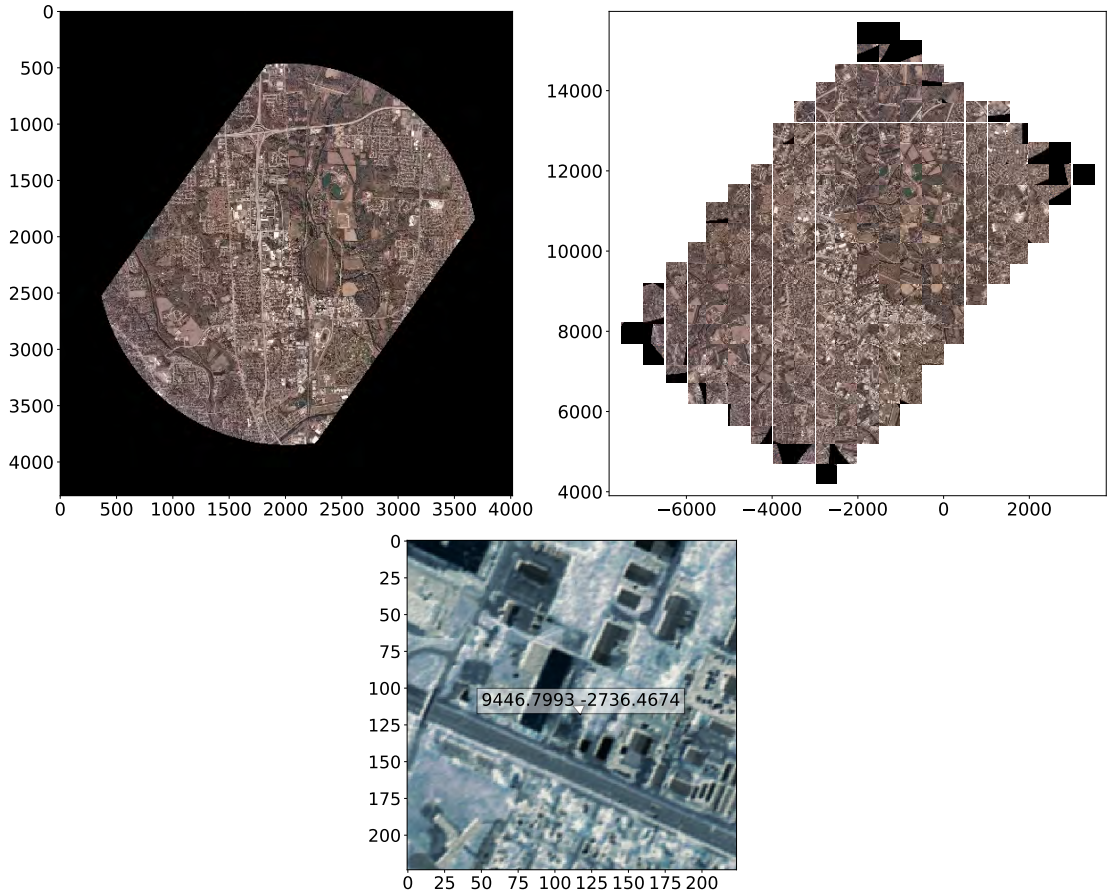

Figure 13. The top left is the original image and falls completely within the Dayton, Ohio bounding box. The top right image has been cut up into 253 sections with a random rotation applied to each section. The bottom image is the output of the process: a sample image with center-point coordinates, which corresponds to one of the small squares in the top right image.

3.1.4 Additional Training Enhancements

Since the goal of this research is to compare various training and performance characteristics of CNNs, it was determined to minimize variability by building a common training and test set. 100,000 input images with their corresponding location coordinate labels were established for the training set. The testing dataset contained 20,000 images and coordinate lables. The network will be trained each epoch on 90,000 observations, and validated with 10,000 observations. The testing dataset contains 20,000 images from a separate set of satellite images.

A dataset that provides a high amount of training set variability would be better for training, but training set variability may also present abnormalities in network learning, causing some networks to learn at rates that are different from others. Therefore, to control variability, additional training methodologies to better train the networks were not implemented. To aid in developing a robustly trained network for future efforts for the aerial visual localization problem, methods such as overlapping circle radius's, multiple orientations of images, randomized datasets within the training process, and various skew or lighting distortions should be implemented.

3.2 System Architecture

In this section the programming language, machine learning infrastructure, hardware, and CNN design is discussed. The Python language was used with multiple specialized packages applying specifically to this dataset. Keras deep learning API, with Tensorflow backend was used to develop the CNN framework. Government Amazon Web Services (AWS) instances was used for computation because of their incorporation of multiple Graphics Processing Units (GPUs). The network design focuses on building a consistant CNN infrastructure to be modified to compare various parameters.

3.2.1 Programing Infrastructure

The language for this study is Python version 3.6. Python is an open-source, high level object-oriented programming language. There is a large and expanding library of packages to aid in programming[80]. Python is the most popular programing language for machine learning[81, 76]. This language was selected primarily for its ease of use and its compatibility with the Keras framework[76]. Because of this, there is a large selection of packages that have been previously created to aid in this specific task.

The algorithms for this thesis utilize standard and specialized Python packages for various tasks. Some of the specialized packages and modules include *mercantile*, which returns bounding coordinates and quadkey (grid location based on zoom) conversions. *Shapely* is used to find bound interactions between the satellite images, the Dayton bounding box, and the individual sub-images. *Affine* is used to manage satellite affine transformation matrices, and *Pyproj* utilizes these tranformations along with the coordinate reference system to return accurate location information within the image. Autonomy and Navigation Technology (ANT) Center and AFRL programming utilities were used to aid in coordinate conversions and data processing. There are also many common Python libraries used such as *numpy, glob, h5py* and many more which are widely used and documented across the Python community.

3.2.2 Machine Learning Platforms

Keras version 2.2.2 was selected as the framework to develop the CNNs. Keras has recently been adopted by TensorFlow as the TensorFlow's high-level API[76]. TensorFlow version 1.10.0 was utilized as the backend engine to develop and run the neural net because it is the largest actively developed backend, and is the most used in the machine learning community[76, 82]. The large community base affords additional benefits in cross collaboration on forums and community channels to aid

in development.

Keras can also be utilized to run on different backends if another infrastructure is required. Keras was developed to be a rapid prototype environment; meaning that with minimal coding fundamental model changes can be implemented[76]. Rapid protoyping is useful, as this study requires multiple changes to analyze various CNN architecture designs.

3.2.3 AWS Instances

Training deep CNN architectures require extensive processing power. As discussed in Section 2.7.1, modern advances in GPU processing have afforded deeper, more sophisticated networks. Government AWS provides advanced cloud services with access to instances with multiple GPUs. Two AWS instances were utilized to provide the capacity to train two networks at the same time. The first instance was a p2.8xlarge which supplied eight NVIDIA K80 GPUs. This instance provided 64Gb GPU memory with 32 virtual Central Processing Unit (CPU)s and 488Gb of RAM. The second was a p3.8xlarge instance that utilized 4 of NVIDIA's advanced V100 Tensor core GPUs which provides 128GB of GPU memory also with 32 virtual CPUs and 244Gb of RAM. Both operating systems utilized a Linux operation system version 4.4.0-140-generic x86 64. Both systems run NVIDIA CUDA toolkit which unleashes the full power of the GPU CUDA cores and in the NVIDIA V100 the tensor cores as well. Because the NVIDIA V100 is specially designed for complex Artificial Intelligence (AI) problems, the p3.8xlarge outperformed the p2.8xlarge in training time and was able to train deeper networks that the p2.8xlarge could not train.

3.2.4 Network Design

This study focuses on network architecture and hyperparameter performance which requires a root setup that minimizes variability but allow the different approaches described in Chapter 2 to be swapped in and out. Keras functional API was utilized to build the network, which is illustrated in Figure 14. The input format is a $224 \times 224 \times 3$ image. The input leads into one of the benchmark deep learning models discussed in Section 2.7. The model is selected from Keras's pre-built applications. The models available for selection are: Xception, VGG16, VGG19, ResNet50, InceptionResNetV2, MobileNet, MobileNetV2, DenseNet121, DenseNet169, DenseNet201, NASNetMobile, and NASNetLarge. The finishing layers from the benchmark model were removed to lead into one of the finishing layers discussed in Section 2.6: flatten, global average pooling, or global max pooling. To increase connectivity, an additional dense layer with 1024 neurons and a ReLU activation was implemented post finishing layer and prior to the output layer. The ouput layer was a two neuron (North, East) dense layer with a linear activation.

The weights will be initialized differently throughout the study with either the pretrained *imagenet* weights from the Keras application, or one of the weight initializations discussed in 2.4: Glorot normal, Glorot uniform, or orthogonal.

After the model was created, a loop went through each layer and updated the weights of convolution and dense layers to the correct weight initializations. Before compilation the model was optimized using the Keras multi GPU model utility. The model is optimized for eight GPUs when run on the p2.8xlarge AWS instance and four on the p3.8xlarge AWS instance. The multi GPU model utility replicates the model on the different GPUs, divides the inputs into sub-batches, executes the models on their dedicated GPU, and then concatenates the results on the CPU into one big batch. Because the GPU technology and processing is different there is a potential

for some discrepancy between instances.

Mean squared error was utilized for the loss to emphasize the error when further from the target. Mean absolute error is used as a training metric for better readability of the location error. Networks were trained using one of the optimizers discussed in Section 2.5: Adam, Adadelta, or RMS Prop. The learning rate is 0.001 for Adam and RMS Prop. This rate was selected because it is small enough to not dramatically over-step minimums, and not too small that training would be lengthy and difficult. Adadelta learning rate was set to 1.0 because the optimizer continually updates the learning rate as training progresses. Weight decay is not used in either of the studies, but would be beneficial for additional training techniques if used with Adam and RMS Prop optimizers.

3.3 Hyper-parameter Comparison

The hyper-parameter comparison addresses three specific hyper-parameters to determine those that will perform better with this dataset. The parameters are network weight intializations, optimizer functions, and finishing layers. There have been significant advances in these hyper-parameters, and this section discusses the methodology to test the performance of certain parameters on the aerial visual localization dataset.

A default configuration was established, and each hyper-parameter was varied and then compared to the base configuration. The Inception V3 model is the base CNN model for this study, and each model was trained for 50 epochs. The best performing hyper-parameters is then combined and trained for 150 epochs along with the default model. Each network's training error rate is then analyzed, and after training, its performance was evaluated on the testing dataset.

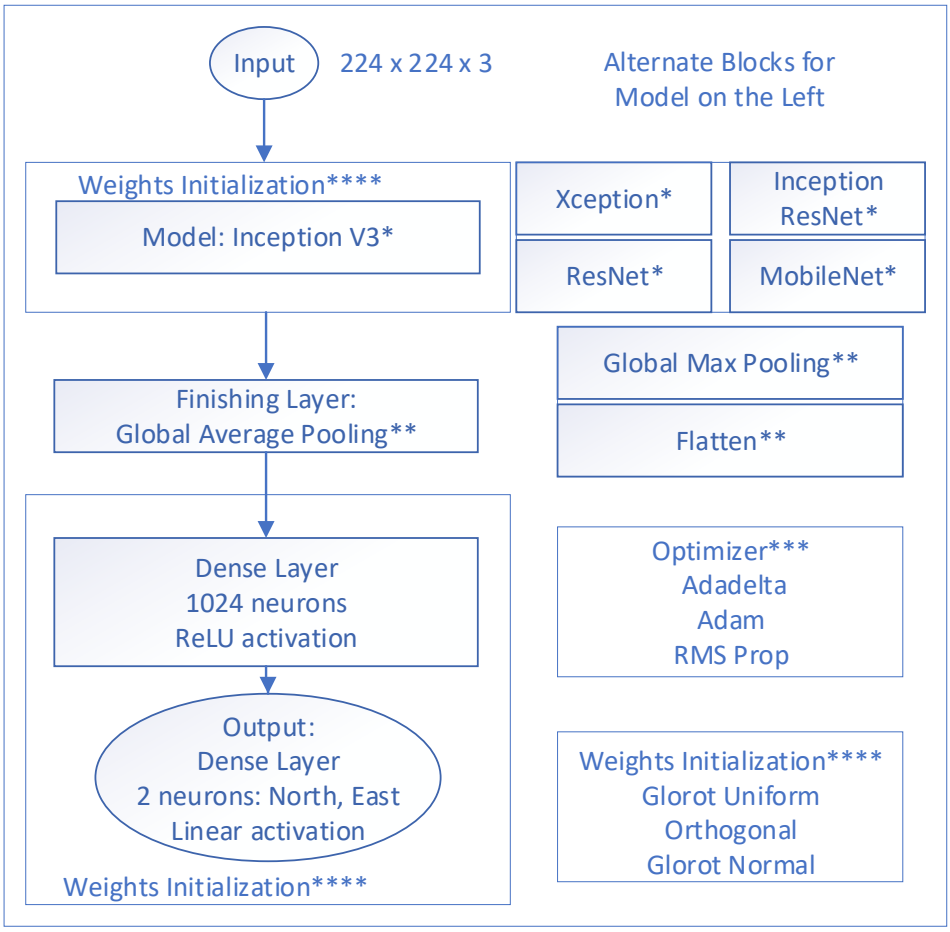

Figure 14. The variables are annotated with a*. The defaults for the hyperparameter comparison are: Inception V3, global average pooling, Adadelta optimizer, and Glorot Normal weight initializations.

3.3.1 Parameters to review

There is extensive customization available for CNNs, especially when it comes to hyper-parameters. For the aerial visual localization task, this research focuses on three hyper-parameters: initialization, optimization, and finishing layers. Figure 15 graphically represents the relationship between weight initializers, optimizers and finishing layers. While Keras application models have pretrained weight sets, when designing custom networks or customizing networks to specific problems, the right initialization can save training time as discussed in Section 2.4. This research will compare three weight initializers: Orthogonal, Glorot Normal, and Glorot Uniform.

Optimizers, as discussed in Section 2.5, have come a long way to advance CNNs into the powerhouse they are today. The development of additional optimizers has also created more choices, and the three to compare in this study are RMS prop, Adam, and Adadelta. Finally, while classification networks enjoy global average pooling as the finishing layer[70, 26, 27], aerial visual localization is a regression problem. To determine the correct finishing layer, flatten, global average pooling, and global max pooling were reviewed.

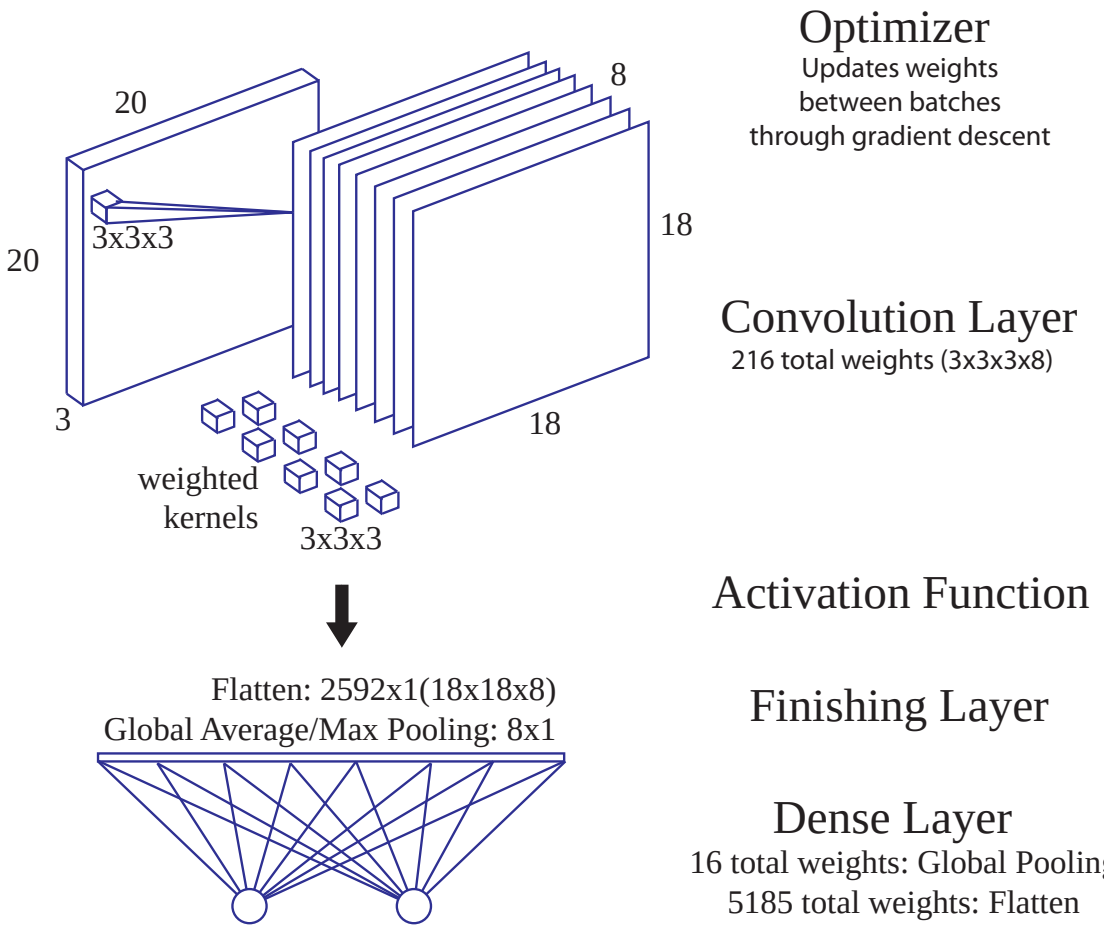

Figure 15. This very simple CNN illustrates the input's relationship between the weight initialization, optimizers and finishing layers. The convolution kernel and dense neuron weights are initialized prior to training. The optimizer updates the weights as training occurs. The finishing layer not only changes the dimensionality in preparation for the output, but affects the balance between network size and information preservation.

3.3.2 Default Configuration

An exhaustive study of the interactions between each of the initializers, optimizers and finishing layers could be beneficial, but due to the training time and instance costs, this study would exceed alloted time and resources. A baseline configuration has been established, and one hyper-parameter is to be varied. The baseline setup includes a Glorot uniform kernel initializer for the convolutional and dense layers. The default optimizer is Adadelta. The default finishing layer, prior to the fully connected layers, is global average pooling.

The InceptionV3 was selected from the prebuilt Keras model applications for all the hyper-parameter comparison tests. The InceptionsV3 architecture was selected due to the architecture reaching state of the art performance with less computational complexity than VGGNet[67]. The use of directed acyclic graphs of layers allows the network to add model complexity to learn spatial and channel-wise features more efficiently than learning them jointly[83]. Furthermore, since InceptionV3 has demonstrated high quality results with low receptive fields, and the images contained in this dataset having very small characteristic features, this network appeared adequate to represent the dataset[67].

3.3.3 Comparison Methodology

Each model was trained for 50 epochs. This amount of training will not produce a fully trained network, but 50 epochs provides adequate training for the network to provide adequate results, illustrate training trends, and validate if the CNN is learning. 50 epochs is also short enough to allow modifications in experimental design without losing months in training time. The Adadelta optimizer could be at a disadvantage with only 50 epochs, since it begins with a poor learning rated and the gradient descent scheme updates the learning rate while training.

The top performing hyper-parameters are combined for a super model and trained to evaluate if a combination of them can improve training and performance. Finally this supermodel and the default configuration was trained for 150 epochs and evaluated based on performance differences. This will determine if Adadelta's performance improves with additional epochs (because it is the default optimizer), and if the advantages of picking the high performers will translate into better performing networks.

The training was measured in two ways: how well the network learns the training dataset, and how well can it generalize the information measured through validation during training. The training rates for each of the initializers, optimizers, and finishing layers were compared with one another in training and validation performance. While final performance is the highest priority, high volatility in the validation set could indicate performance that might not be repeatable. Early in training, a network usually has little reliability, so an emphasis will be placed on middle and later training.

Training advantages can lead to better CNNs in less time, and model performance is essential. Weight initializations have greater importance during earlier training epochs, because the optimizer's feedback overwrites the initializations though training as the network becomes fully trained. Finishing layers could have applicability to both training and performance, and optimizers definitely affect both training and performance.

To measure performance, each of the networks were loaded and their performance evaluated with the test dataset discussed in Section 3.1. The Mean absolute errors of the test set are compared across the various models. The geographic positions of the best and worst errors were analyzed to see patterns and benefits of some networks over others. Finally, images of the top network errors are analyzed visually to determine if image conditions such as cloud cover etc. could be affecting the network.

3.4 CNN Model Architecture Comparison

The model architecture comparison takes industry leading networks and trains them on the aerial visual localization dataset to determine the best performing networks. Each network brings specific advantages in design. Each CNN will be trained with both untrained weight initializations and weights pretrained on the Imagenet[19] dataset. The architectures will be analyzed through their performance, training and network shortcomings.

3.4.1 Models to Review

There are some very innovative CNN model architectures discussed in Section 2.7. Each of these models brings specific advances to the table in terms of problem solving. While there are winners in performance on the Imagenet[19] dataset, this application is not a classification problem on that dataset. The question is this: which model is the best at performing the task of visual aerial localization?

MobileNet V2 has an advantage in being much smaller than all of the others, and if it's performance is comparable to its peers, then it would be an excellent choice especially for smaller UAVs. Inception V3 is much larger than mobileNet V2, but with its directed acyclic graphs, it could perform well on this dataset. ResNet 50 is comparable to Inception V3 in network size, and residual connections could provide an advantage. Xception is slightly smaller, yet comparable in size to Inception V3, and its dependence on depth-wise separable convolution could be an advantage. DenseNet 201 is slightly smaller than Xception and having connections throughout the network could provide excellent performance on this dataset.

3.4.2 Default Settings

The network design described in Section 3.2.4 was used for this comparison. The optimizers and finishing layers remained static. The Adam optimizer and flatten finishing layer was used to train all of the networks. The optimizer and finishing layer were both selected because their performance during the hyper-parameter comparison indicated they perform well on this dataset. Government AWS p2.8xlarge and p3.8xlarge instances were used, so various networks were optimized for either four or eight GPUs as discussed in Section 3.2.3. This means that different networks were trained on different computers.

3.4.3 Model Comparison

To understand the difference in the network performance each CNN was trained for 150 epochs. This amount of training provided sufficient time for networks to overcome instability and demonstrate their ability to learn the information. Each network will be trained two times one with the pretrained imagenet weights, and one with a Glorot Uniform weight initialization. This is done to demonstrate if there is an advantage of utilizing pretrained networks as well as the ability to compare the networks with custom designs that will not have access to pretrained weights.

Each model was compared during training by how well the network learns the dataset, and how well the network generalizes the data. The loss function for learning was mean squared error. Mean Absolute Error (MAE) is used as a training and validation metric. The rates are compared across models by category, and with/without imagenet weights. The difference in each model's performance with imagenet weights vs untrained weights was analyzed. Validation volatility was also be examined.

Network performance was measured based on each network's performance on the test dataset discussed in Section 3.1. The mean absolute error is compared across

each of the networks for both pretrained, and untrained weight initializations. Geographic positions of best and worst errors was illustrated along with error locations to determine trends in dataset errors and variations between where the networks excel. High and low error images were analyzed to visually assess network performance on the dataset.

3.5 Summary

This chapter discussed the techniques and methods used for the experiments in this study. The dataset section covers the in-depth origin and formatting of the satellite images to represent an appropriate aerial dataset. It also discusses dataset formatting techniques for the various CNNs. System Architecture details the programming structures, the machine learning architectures, and CNN designs specific to this thesis. The Hyper-parameter Comparison provided the procedure for comparing nine hyper-parameters. CNN Model Architecture Comparison described the process for comparing seven innovative CNN models. Finally, Custom Loss Development discussed a loss specifically designed to integrate the results of a network into an algorithm with IMU data to provide a more accurate location.

IV. Results

This chapter discusses the training and testing results for the study outlined in Chapter III. Section 4.1 discusses essential details regarding the dataset, and training requirements to complete the study. The Hyper-parameter Comparison discussed in Section 4.2 had diverse training and testing performance. The three hyper-parameters were optimizers, finishing layers, and weight initializers. The convolutional neural network (CNN) Model Comparison in Section 4.3 analyzed five models: MobileNet V2, Inception V3, ResNet 50, Xception, and DenseNet 201.

4.1 Resources

Even with the multiple Graphics Processing Units (GPUs) supplied in both the p2.8xlarge and p3.8xlarge instances, training CNNs was a lengthy process. Running multiple processes while training a CNN model would drain resources and stall or cancel the training. Since many of the models took more than 20 hours to train, processing only one model at a time per instance became a limiting factor.

4.1.1 Dataset

The dataset was built using a collection of satellite images that encompassed in some fashion the bounding area over Dayton Ohio. There are 100,000 images in the training dataset, as described in Section 3.1. The training dataset location coordinates are shown in Figure 16. The distribution is fairly balanced, but the amount of images with coordinates in the corners of the Dayton bounding box are significantly higher than those in the center. This imbalance of data could skew the training by over emphasizing corners vs the area as a whole.

The testing dataset, also described in Section 3.1, contains 20,000 images. The

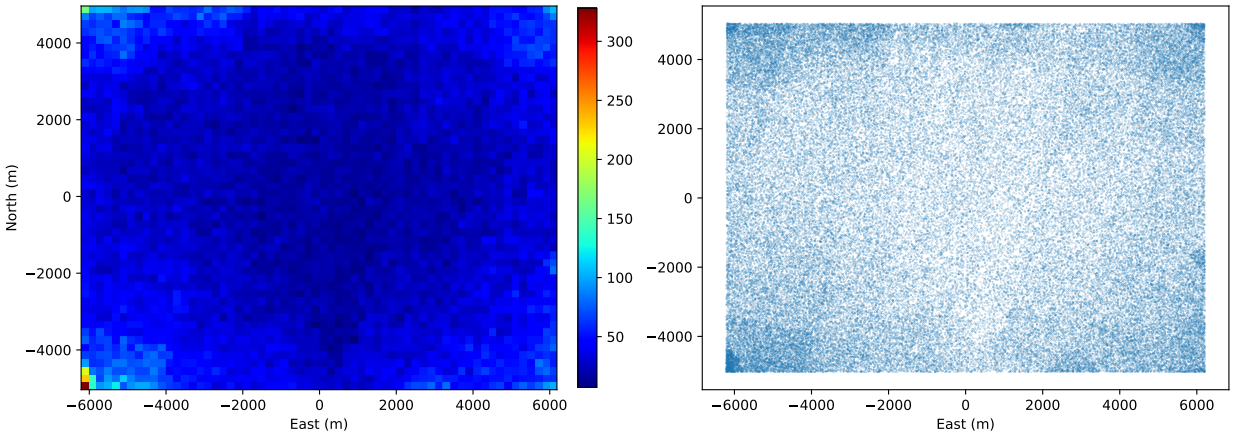

Figure 16. Both charts are based off the 100,000 locations of the dataset points in the training dataset. The histogram on the left displays the density of images for every 200^2 meter block of the Dayton bounding box. The blue dots on the right display the center locations of each image in the dataset against a white background.

testing dataset locations are similar to the training dataset with respect to their location density. This phenomenon is understandable because both datasets come from Planet Labs Inc. satellites with an orbit of the Dayton Ohio area. Because the image location distribution is similar in both datasets, future development in dataset location distribution could provide a more robust testing and training environment.

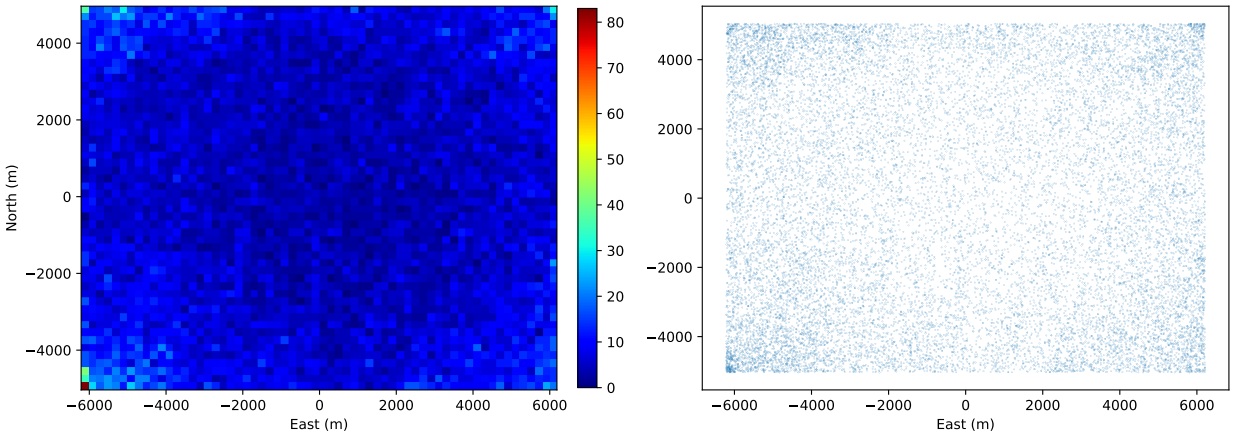

Figure 17. Both charts are based off the 20,000 locations of the dataset points in the testing dataset. The histogram on the left displays the density of images for every 200^2 meter block of the Dayton bounding box. The blue dots on the right display the center locations of each image in the test dataset against a white background.

4.1.2 Equipment

Training time was a significant limiting factor. Even with the significant processing power of the Amazon Web Services (AWS) instances, as described in Section 3.2, it took approximately 350 to 2000 seconds per epoch to train an epoch on the p3.8xlarge instance. Training time took approximately 500 to 3500 seconds per epoch on p2.8xlarge instance. Training the default model in the hyper-parameter comparison for 150 epochs on the p3.8xlarge instance took 16 hours. To train the super-model in the hyper-parameter comparison for 150 epochs on the same instance took 21 hours. The increased training time was due to the additional parameters of the flattening layer vs the global average pooling finishing layer.

The models in the CNN comparison varied significantly in the required training time. Due to the various network sizes and capabilities of the AWS instances, the batch size required adjusting depending on the instance and the model size. The batch size was initially set, then taylored if the instance ran out of memory during training. Any background processes also affected memory usage, so network size was not the only factor in determining batch size. The list of model parameter size and the batch size is listed in Table 2. The Inception-ResNet had significant learning issues when trained on the p2.8xlarge, which resulted in multiple out of memory errors and failure of the network to learn. When ran on the p3.8xlarge instance, no similar issues were observed.

4.2 Hyper-parameter Analysis

The performance during training and testing was diverse. The studied hyper-parameters were optimizers, finishing layers, and weight initializations. The optimizers analyzed were Adadelta, Adam and RMS prop. The finishing layers were global average pooling, flatten and global max pooling. The weight initializations were Glo-

Model Parameters and Batch Size				
Model Type	Total Parameters	Trainable Parameters	Training Batch Size	AWS Instance
Hyper-parameter Default	23,903,010	23,868,578	128	p3.8xlarge
Hyper-parameter Super-model	74,234,658	74,200,226	64	p3.8xlarge
Inception V3	74,234,658	74,200,226	64	p2.8xlarge
DenseNet 201	114,662,978	114,433,922	64	p3.8xlarge
MobileNet V2	66,486,338	66,452,226	64	p2.8xlarge
ResNet 50	126,351,234	126,298,114	64	p2.8xlarge
Inception-ResNet V2	93,661,410	93,600,866	32	p3.8xlarge
Xception	123,625,002	123,570,474	64	p3.8xlarge

Table 1. This table illustrates the batch size in relationship to the trainable parameters and AWS Instance.

rot normal, orthogonal, and Glorot uniform. The default model was comprised of an Adadelta optimizer, a global average pooling finishing layer, and a Glorot uniform weight initialization.

All hyper-parameters were varied one at a time and tested against their respective classes (optimizers, finishing layers, and weight initializations). Each model was trained for 50 epochs. The best performance during training was observed in networks containing the RMS prop optimizer, the flatten finishing layer, and orthogonal weight initializer. This selection was combined to develop a super-model to be trained for 150 epochs and compared to the default model.

The performance of the hyper-parameter comparison on the test dataset did not mirror the performance of the best hyper-parameters during training and validation time. The performance of each model on the test set after 50 epochs of training analyzed and discussed in detail in Section 4.2.2 below.

The supermodel outperformed the default model in the 150 epochs of training and validation. The testing performance of the super-model was also a significant improvement over the default model after being trained for 150 epochs.

4.2.1 Training

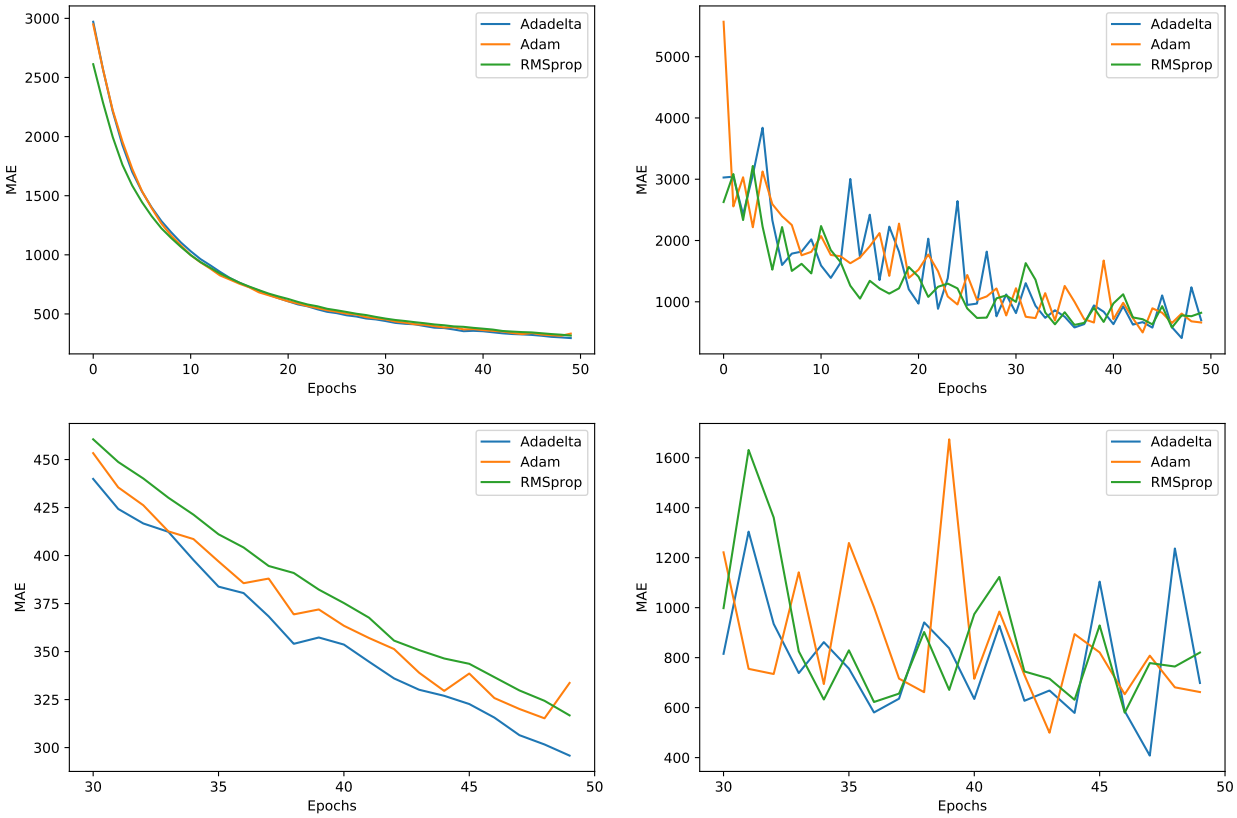

Figure 18. The charts on the top are training over the full 50 epochs, and the charts on the bottom are a zoom of the epochs 30 to 50. The charts on the left are the training errors of all three optimizers Adadelta, Adam, and RMS prop. The charts on the right are the validation error rates for each epoch.

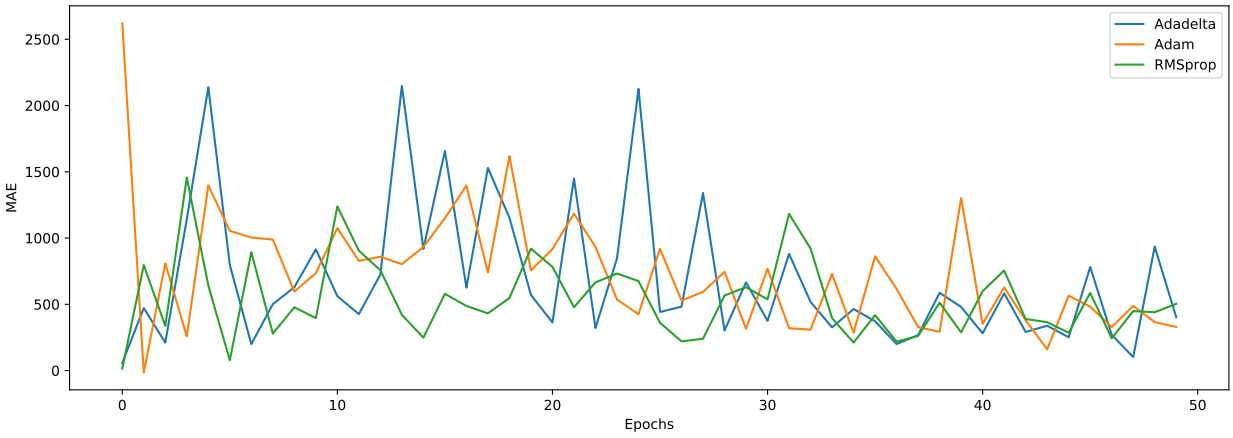

Figure 19. This chart shows the validation error minus the training error per epoch for the optimizers Adadelta, Adam, and RMS prop.

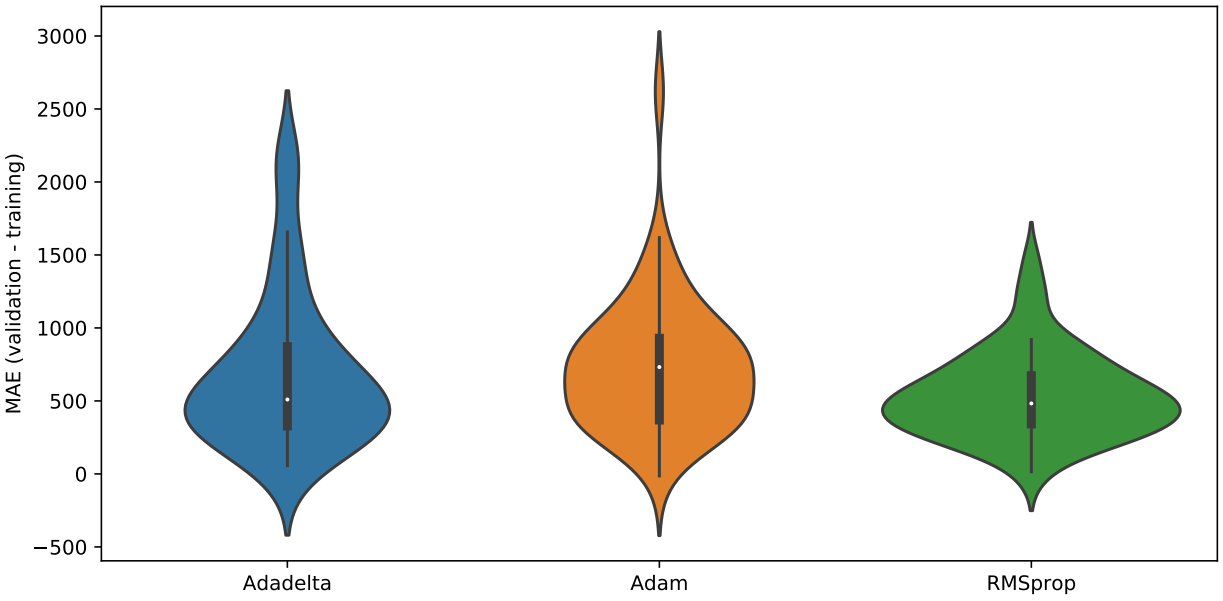

Figure 20. This is a violin plot of the validation minus training data as displayed in figure 19 for the optimizers Adadelta, Adam, and RMS prop.

Optimizers The three optimizers analyzed in the hyper-parameter study were Adadelta (which was the default model), Adam, and Root Mean Squared (RMS) propagation. All three optimizers performed similarly well with respect to the training error as seen on the left hand side of Figure 18. The RMS prop training error was lower in the earliest epochs, but was surpassed by Adadelta and Adam in later epochs. Adadelta had a higher variability in the validation data in the earlier epochs. This is expected because of Adadelta's algorithm as discussed in Section 2.5.

The validation error minus the training error was used as a metric to analyze the model's ability to generalize the data learned through training. The charts of this metric is displayed in Figures 19 and 20. The difference between the training curve and the validation displays the network's ability not just to memorize training data, but to translate the data into learned patterns. Figure 19 shows the difficulty the Adadelta model had in the earlier epochs to generalize the training data. Adadelta had significanly higher variability and volatility earlier on, but toward the later epochs the Adadelta model had less volatility than the Adam model. This pat-

tern is displayed more succinctly in Figure 20. This violin plot displays Adadelta's 95% confidence interval as the highest of the three, yet the interquartile range and median are both less than the Adam model. The Adam model appears in Figure 19 to perform fairly well, but the violin plot reveals the lackluster performance of the Adam model compared to the RMS prop model. RMS prop appears to perform extremely well in both Figures 19 and 20. The confidence interval and interquartile range are tight and much smaller than the rest, portraying the ability to generalize the information well with little volatility. The median error for RMS prop is slightly higher than Adadelta, however considering all aspects, it appears that RMS prop performed the best during the first 50 epochs of training when compared to the other two optimizers Adadelta and Adam.

Finishing Layers The three finishing layers were global average pooling (the default finishing layer), flatten, and global max pooling. All three models' training and validation error charts are in Figure 21. It is not surprising that global average pooling had the best training error curve, with it's popularity as a finishing layer, as described in Section 2.6. The validation error did not appear to follow a similar trajectory as the training error.

The validation error minus the training error metric in Figures 22 and 23 tell a much different story about network generalization. The volatility of flatten finishing layer is much less than that of global average pooling and global max pooling. Flatten does add significant bulk in training parameters to the network, but it had the tightest 95% confidence interval and interquartile range. The validation errors were also on average closer to the training errors than the global average and global max pooling. Since flattening could add significant network size (depending on CNN output) as compared to the other, a possibility for a lighter network could be to combine global average and global max pooling. This combination would still be much lighter than

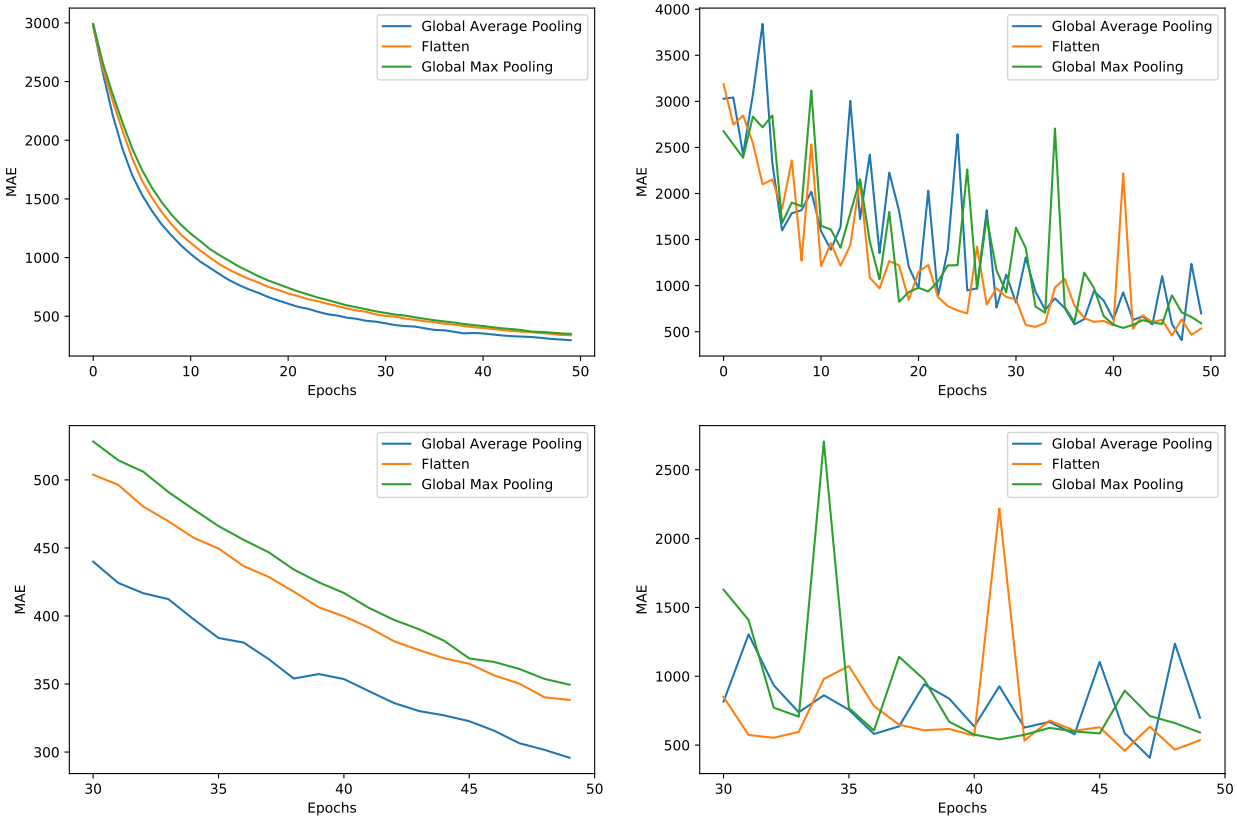

Figure 21. The charts on the top are training over the full 50 epochs, and the charts on the bottom are a zoom of the epochs 30 to 50. The charts on the left are the training errors of all three finishing layers: global average pooling, flatten, and global max pooling. The charts on the right are the validation error rates for each epoch.

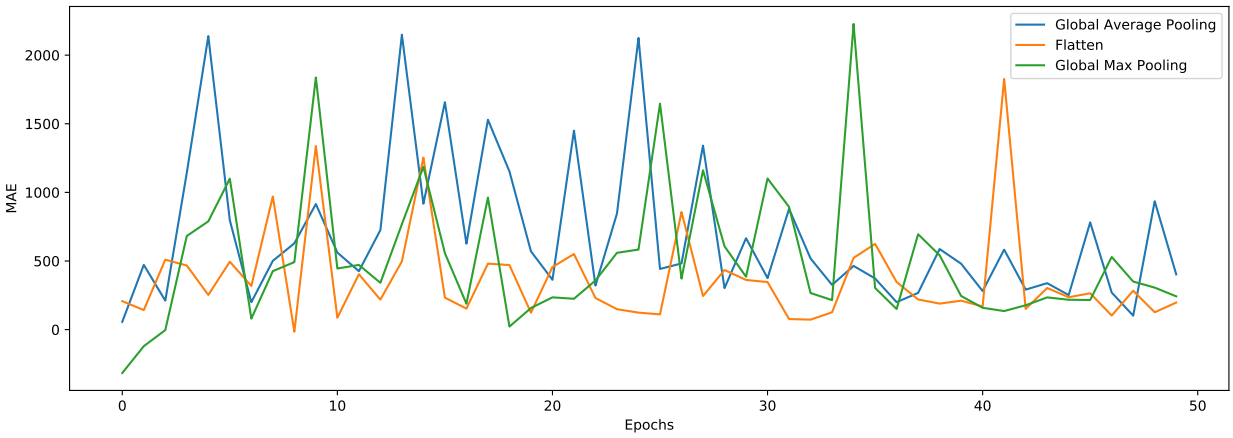

Figure 22. This chart shows the validation error minus the training error per epoch for the finishing layers: global average pooling, flatten, and global max pooling.

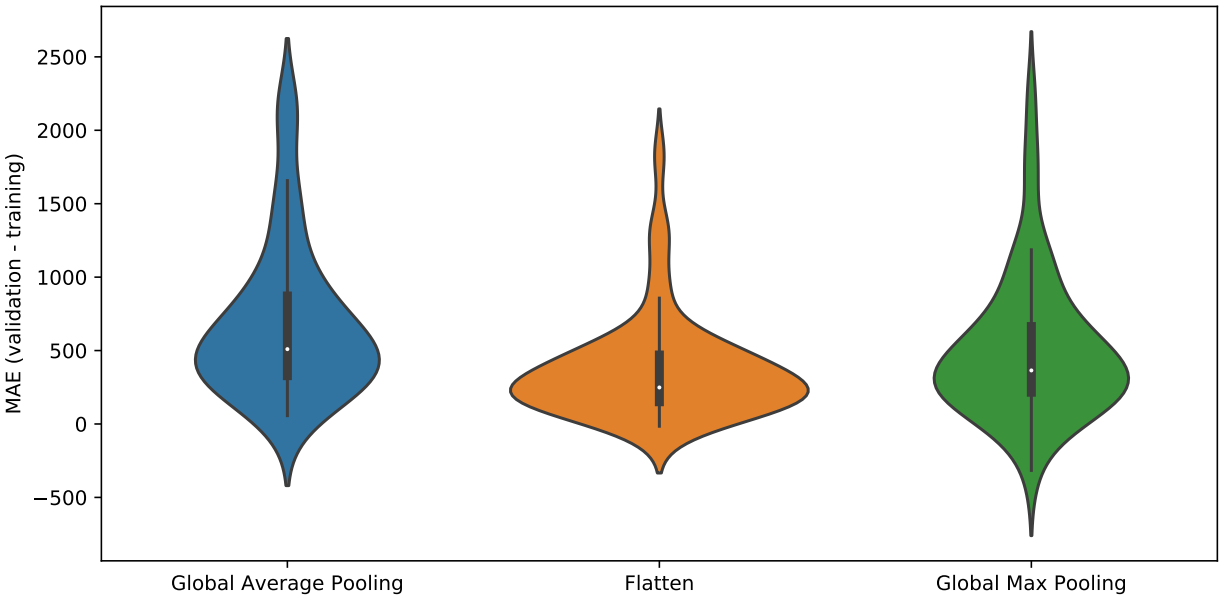

Figure 23. This is a violin plot of the validation minus training data as displayed in Figure 22 for the finishing layers: global average pooling, flatten, and global max pooling.

flatten, but could add additional value in feature detections. It is clear that flatten outperformed both global average pooling and global max pooling for this comparison. However, due to the additional overhead with a flattening finishing layer, performance vs network size tradeoff must be considered for the application.

Weight Initializations The three weight initializers: Glorot Uniform, orthogonal, and Glorot normal had a fairly balanced training error rate. Glorot uniform had a slightly better training error throughout the epochs as seen in Figure 24. Analyzing the validation error rates minus the training error in Figures 25 and 26, there is a significant performance advantage of the orthogonal initialization over Glorot uniform. Glorot normal also performed better than Glorot uniform with a much tighter 95% confidence interval. Orthogonal is the clear winner, with the tightest interquartile range and 95% confidence interval. This emphasizes the ability of orthogonal weight initialization to generalize the data consistently through training.

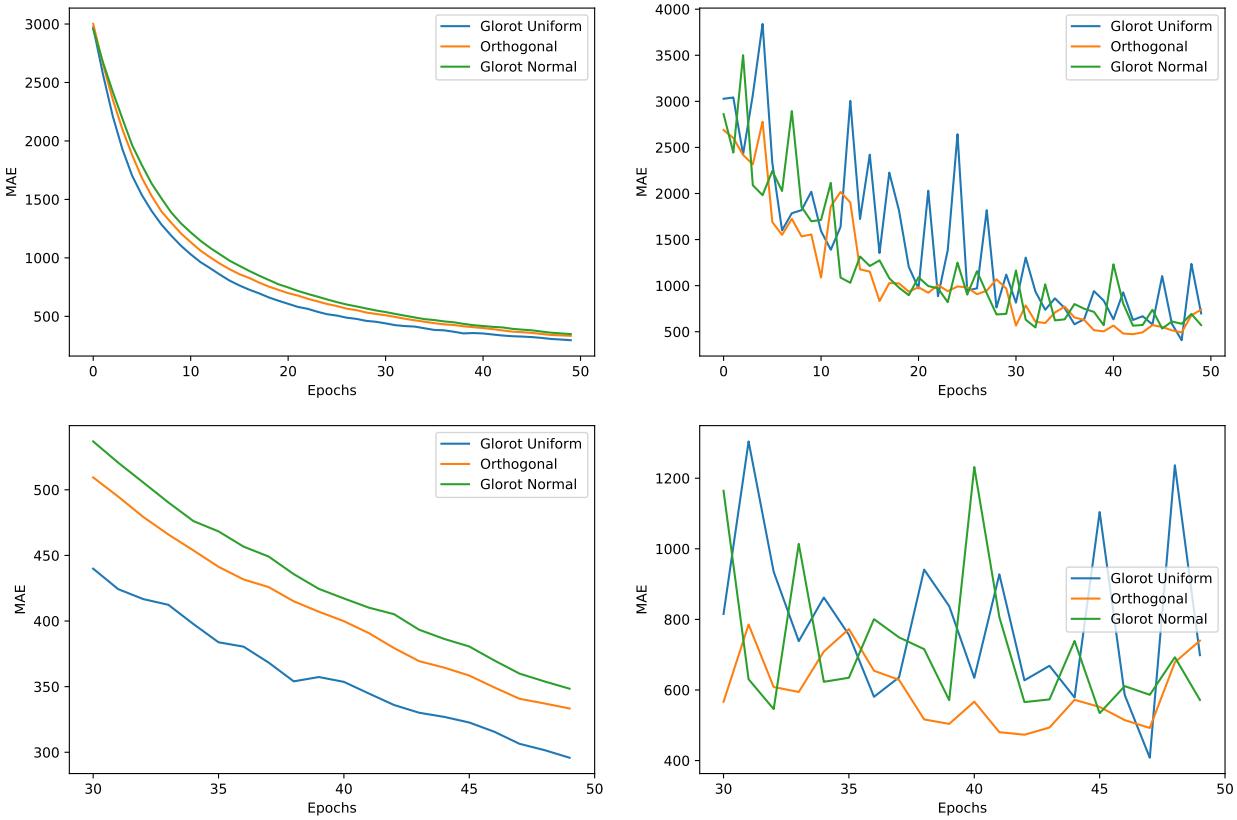

Figure 24. The charts on the top are training over the full 50 epochs, and the charts on the bottom are a zoom of the data epochs 30 to 50. The charts on the left are training errors for all three weight initializers: Glorot uniform, orthogonal, and Glorot normal. The charts on the right are the validation error rates for each initializer over the epochs.

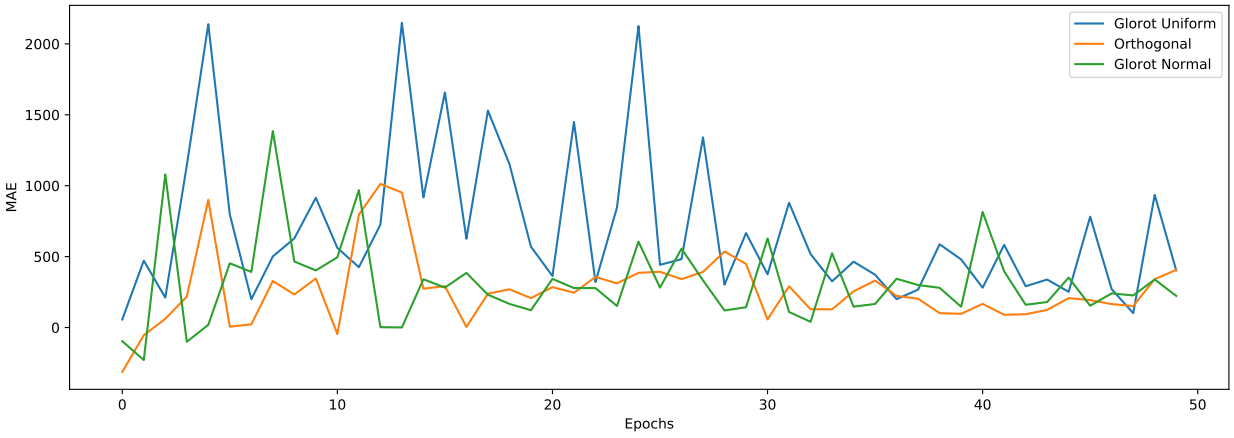

Figure 25. This chart shows the validation error minus the training error per epoch for the weight initializers: Glorot uniform, orthogonal, and Glorot normal.

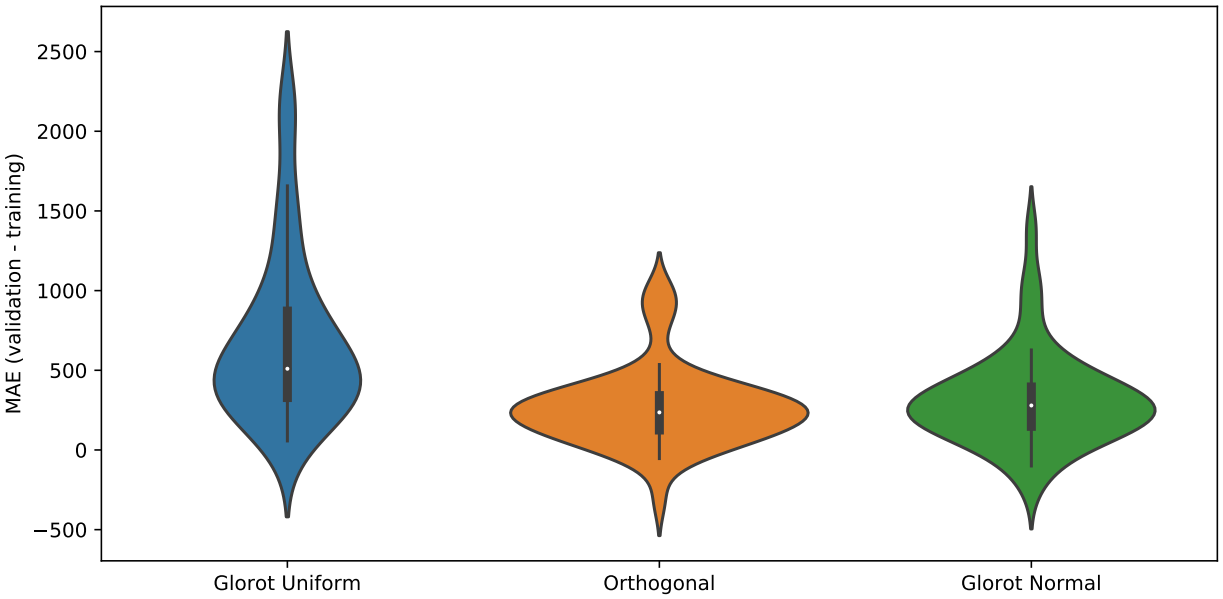

Figure 26. This is a violin plot of the validation minus training data as displayed in Figure 25 for the weight initializers: Glorot uniform, orthogonal, and Glorot normal.

Super Model All of the previous comparison networks were only trained to 50 epochs. This level of training is insufficient to provide an adequate solution for visual aerial navigation, as there is still room for additional training before overfitting. A super-model was devised based on the best performing hyper-parameters to evaluate the benefit of those parameters with adequate training on the dataset. The best performing hyper-parameters for the super model were: RMS prop optimizer, flatten finishing layer, and orthogonal weight initializers. This model will be compared with the default model which has an: Adadelta optimizer, global average pooling finishing layer, and Glorot uniform initializer. The training and validation error can be seen in Figure 27.

In Figure 27 the super-model outperformed the default model. The benefits of the super-model's hyper-parameters are apparent. In Figure 28 and 29 the validation error minus the training error displays the super-model's ability to consistently generalize the training data through validation better than the default model. From the earliest epochs down to the last, the super-model reduces variability and volatil-

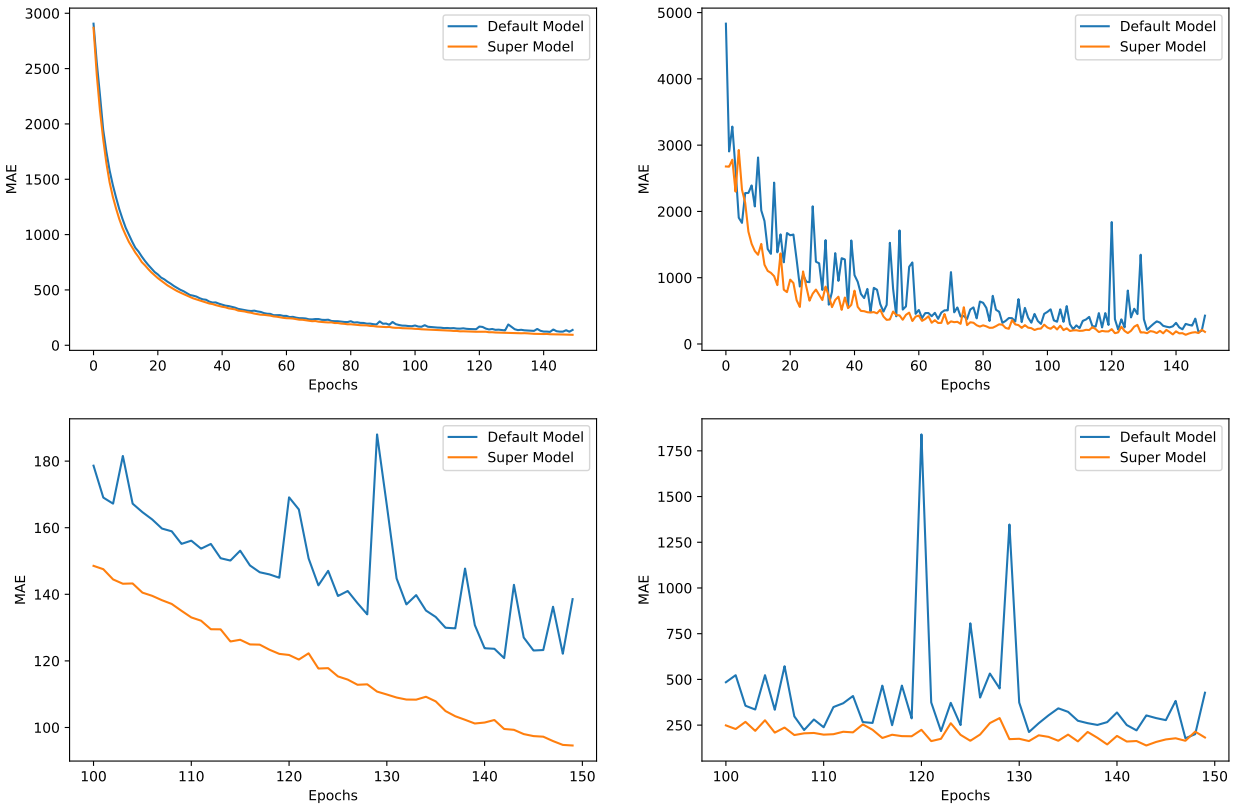

Figure 27. The training and validation error for the default model and the super-model. The charts on the top are training over the full 150 epochs, and the charts on the bottom are a zoom of the data epochs 100 to 150. The left charts are training errors, and the right are the validation error rates for both models over the epochs.

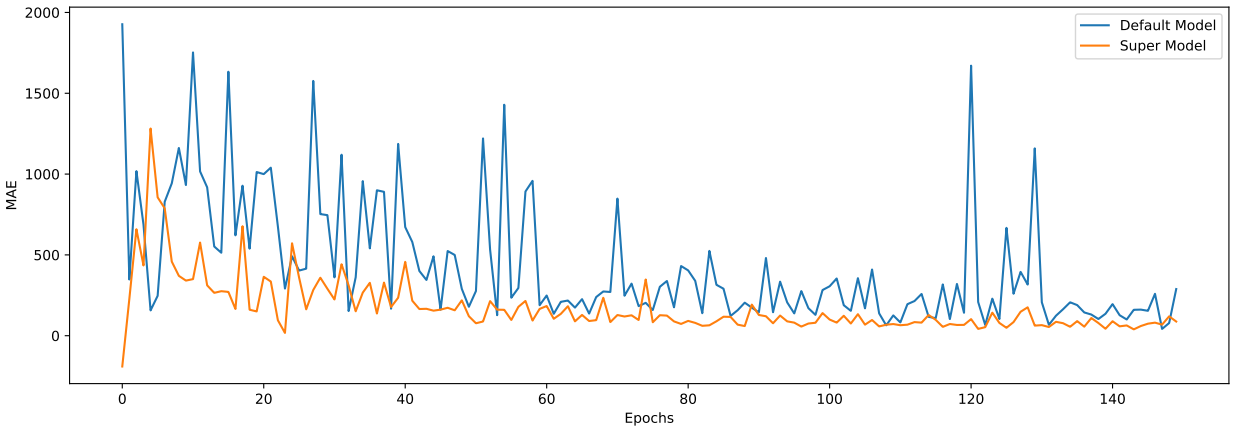

Figure 28. This chart shows the validation error minus the training error per epoch for the default model and the super-model.

ity. Figure 29 emphasizes the dramatic difference between the performance of the two models during training. The 95% confidence interval and the interquartile range

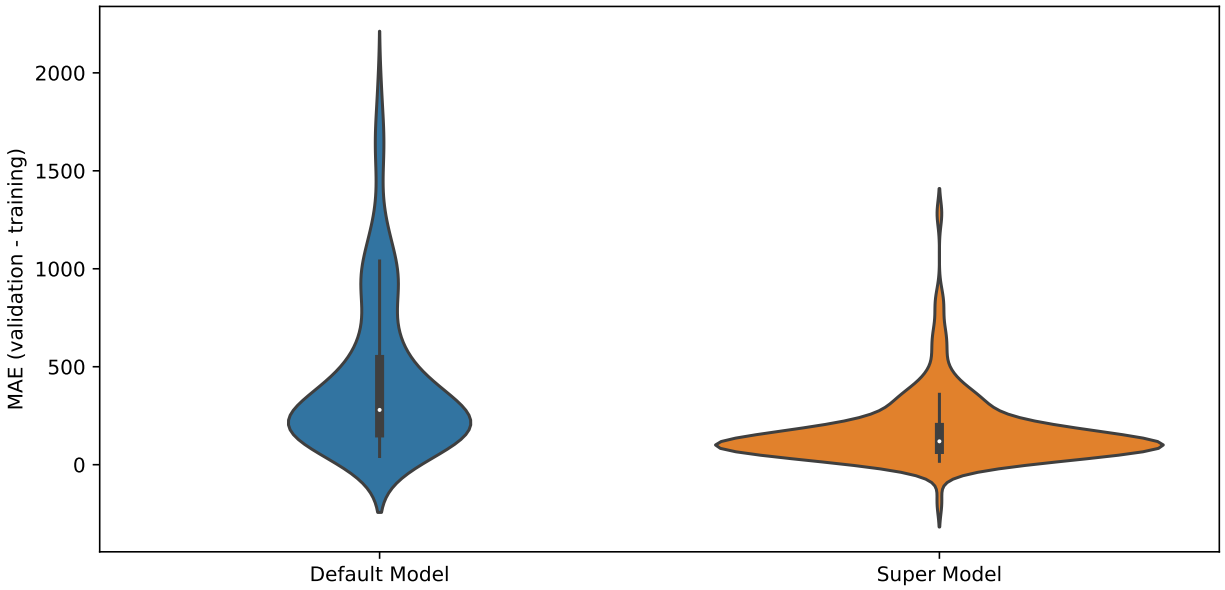

Figure 29. This is a violin plot of the validation minus training data as displayed in Figure 28 for the default model and the super-model.

are significantly tighter containing lower maximums than the default model. These results demonstrate that carefully selecting the appropriate hyper-parameters can greatly affect the networks ability to generalize training data.

4.2.2 Testing

Testing occurred in two stages. Stage one is testing the hyper-parameter comparison models trained over 50 epochs. Stage two is testing the super-model versus the default model after 150 training epochs.

4.2.2.1 Hyper-parameter Testing

The hyper-parameter test results were typically not in-line with the best validation results found in 4.2.1. Training each model in the hyper-parameter comparison for 50 epochs provided extensive data in determining patterns of success in training. Conversely, it was not adequate to fully train the CNNs, as will be discussed below. Figure 30 contains the results of the test set in a side by side comparison between

Table 2. The default model is listed first with Adadelta optimizer, global average pooling finishing layer and Glorot uniform weight initializations. The Adam and RMS Prop were the additional optimizers that were varied. Flatten and Global Max Pooling were the finishing layers, and Orthogonal and Glorot Normal were the additional weight initializers that were tested.

Test Set Frobenius Norm Error after 50 Training Epochs (Meters)				
Model Type	Mean Error	Median	Max Error	Min Error
Default Model	1129.582	883.531	12550.085	7.721
Adam	1049.041	722.754	11748.307	7.788
RMS Prop	1309.568	797.242	12146.415	1.392
Flatten	834.358	578.957	11807.755	4.608
Global Max Pooling	950.918	701.362	11887.215	5.414
Orthogonal	1168.798	730.651	11787.325	4.233
Glorot Normal	910.526	710.760	11074.459	4.989

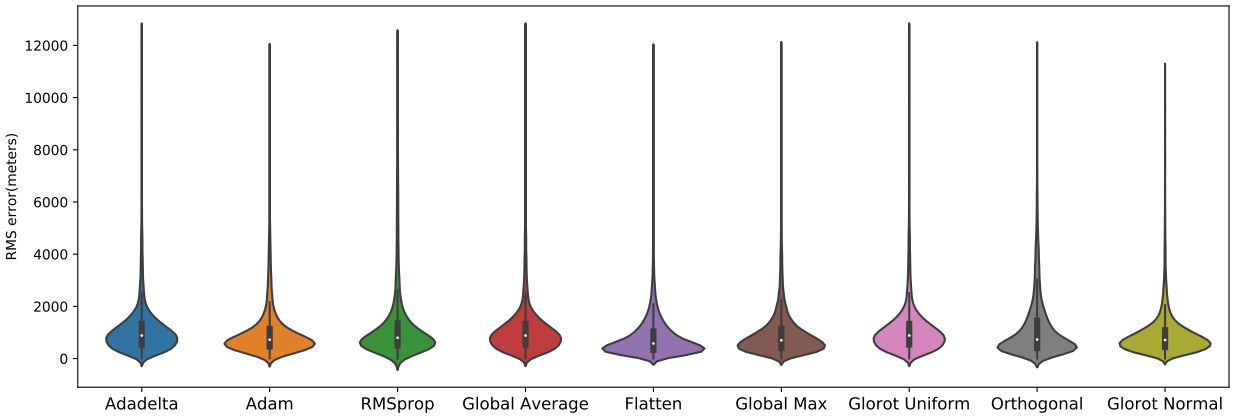

Figure 30. This violin plot displays the RMS error results of the test dataset on all the hyper-parameter comparison models. Each model was trained for 50 epochs.

all the hyper-parameters. It is observationally evident that there is a small level of extreme outliers in every model compared. These extremes are shown in Table 2. Each model has similar values for the maximum errors, but the mean errors are diverse. The model run with a flatten finishing layer was the most successful on the test dataset. The performance of the network with flattening was predicted based on the performance during the training dataset. The other hyper-parameters that were expected to perform the best were actually the worst performing on the test dataset. RMS prop, which was the preferred optimizer, had the worst RMS mean test error out of all the models, and the orthogonal weight initializer had the second worst RMS

mean test error.

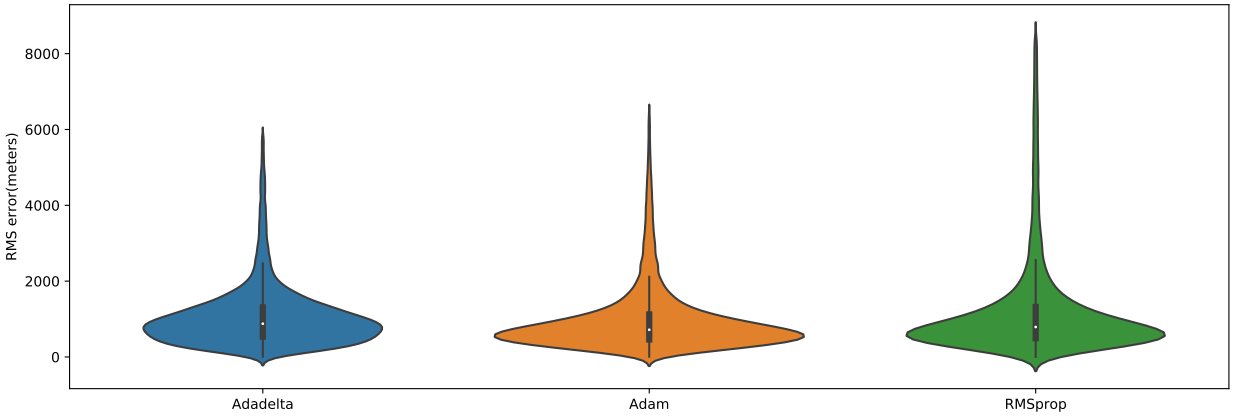

Figure 31. This violin plot displays the RMS error results of the test dataset for the Optimizer comparison models: Adadelta, Adam, and RMS prop. Each model was trained for 50 epochs, and the worst 200 RMS errors were removed for readability.

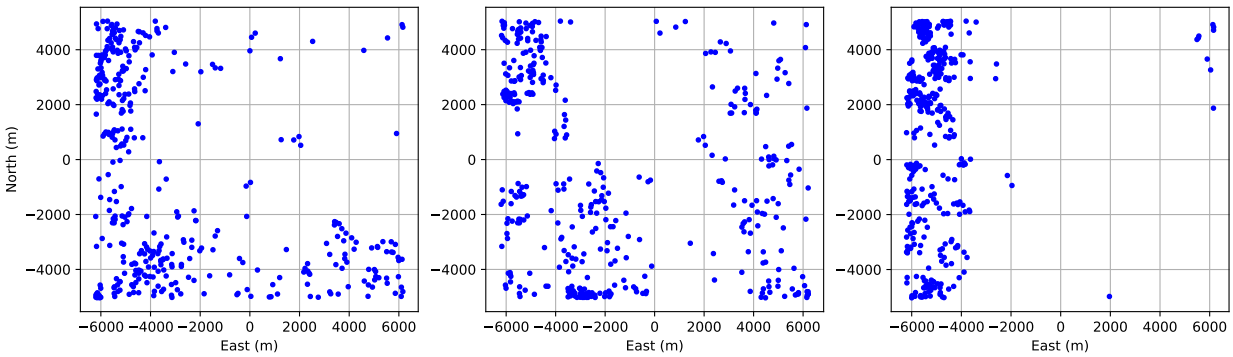

Figure 32. This scatter plot indicates the actual location of the 400 highest errors for the optimizer models: Adadelta, Adam, and RMS prop.

Optimizers The extreme error values made it difficult to observe performance advantages due to the skew in the charts. To enhance the readability, these extremes were clipped in Figure 31. It is observed that the RMS prop's predictions had a much higher rate of error than the other two optimizers, even with the worst 200 errors removed in the figure. It is surprising to see that Adam turns up as the best performer after the lackluster performance during training. This underlines the notion that training errors are a poor prediction of actual model performance.

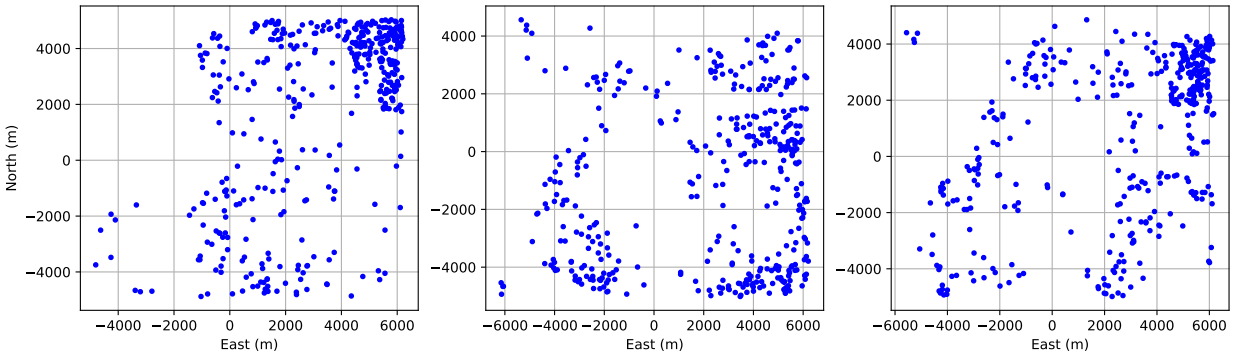

Figure 33. This scatter plot indicates the actual location of the 400 lowest errors for the optimizer models: Adadelta, Adam, and RMS prop.

What sheds further light on the performance of the three optimizers are the results in Figures 32 and 33. Observing the highest error distribution of the three optimizers illustrates that RMS Prop was extremely poor at classifying specific areas of data, and the best classifications in Figure 33 also were geographically localized. The Adam model had the most balanced distribution between the high errors and the low errors. Adadelta also had some regions that performed better and worse. The highest errors for the model using the Adam optimizer follow the dataset distribution described in 4.1.1. It appears that the Adam optimizer was the best at generalizing the training data for testing.

Finishing Layers The performance of the finishing layer models: global average pooling, flatten, and global max pooling, were similar to expected results. Figure 34 contains the violin plot of the RMS error distribution for the three models. With the worst 200 errors removed to enhance readability, it is clear that the model with the flatten finishing layer had an error distribution on the test set that outperformed the other two finishing layers. It could be argued that the advantage of the additional weights created by the flatten would aid to better generalization. Recall that the flatten model has 74,200,226 trainable weights where the other two have 23,868,578.

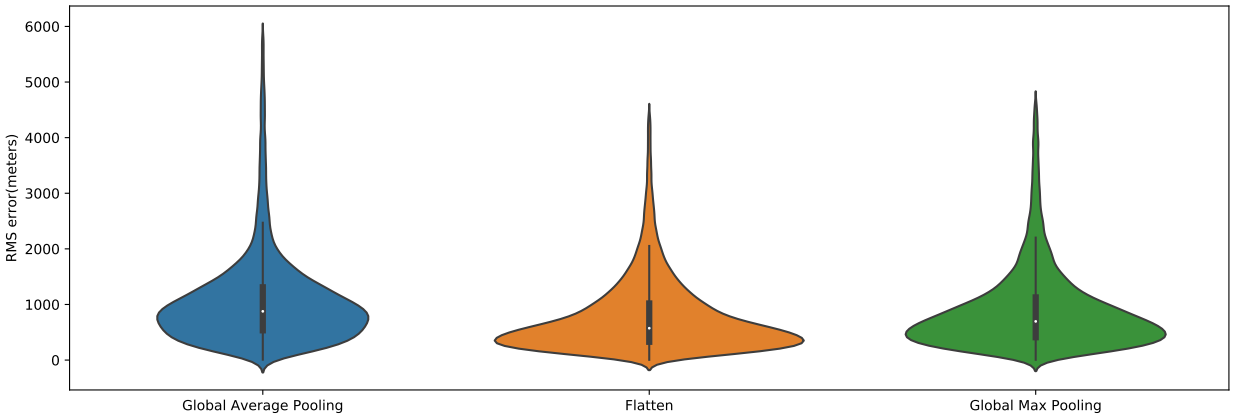

Figure 34. This violin plot displays the RMS error results of the test dataset for the finishing layer comparison models: global average pooling, flatten, and global max pooling. Each model was trained for 50 epochs, and the worst 200 RMS errors were removed for readability.

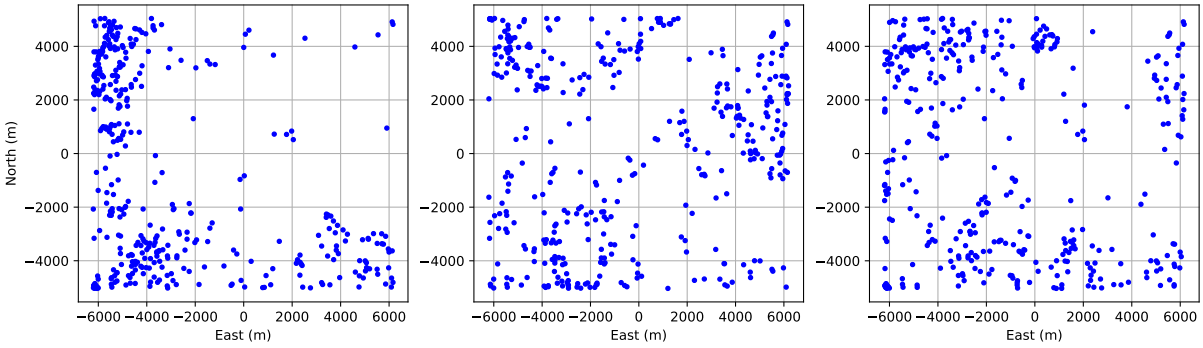

Figure 35. This scatter plot indicates the actual location of the 400 highest errors for the finishing layer models: global average pooling, flatten, and global max pooling.

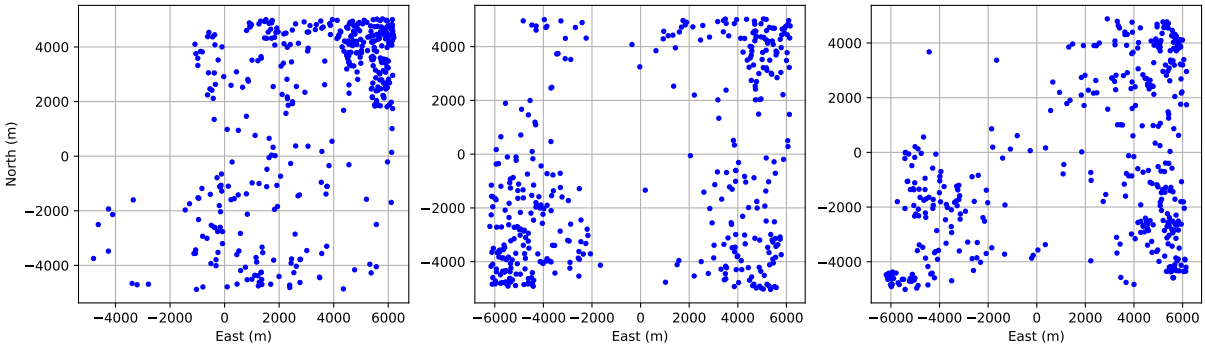

Figure 36. This scatter plot indicates the actual location of the 400 lowest errors for the finishing layer models: global average pooling, flatten, and global max pooling.

The error distributions in Figure 35 depicts that the global average pooling model had higher error rates in the west and south segments of the dataset. The error

distributions for both global max pooling and flatten models did not appear regionalized. The lowest errors, as seen in Figure 36, show the flatten model following a pattern similar to the dataset distribution. The global average pooling and global max pooling models appear to localize their best error rates to a specific location. These results show that the flatten model has a better geographical distribution of the area. This could also be due to the additional trainable parameters provided by the flattening finishing layer.

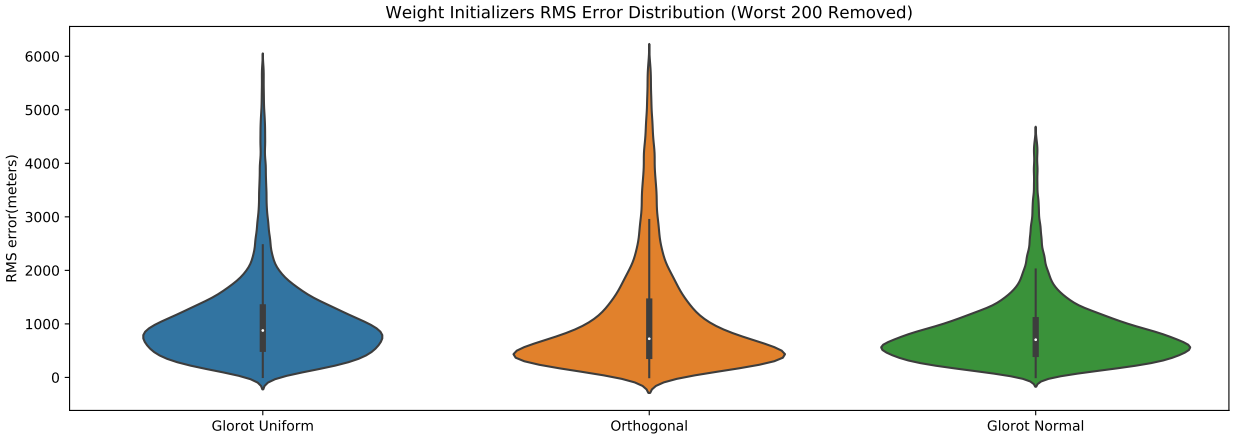

Figure 37. This violin plot displays the RMS error results of the test dataset for the weight initializer comparison models: Glorot uniform, orthogonal, and Glorot normal. Each model was trained for 50 epochs, and the worst 200 RMS errors were removed for readability.

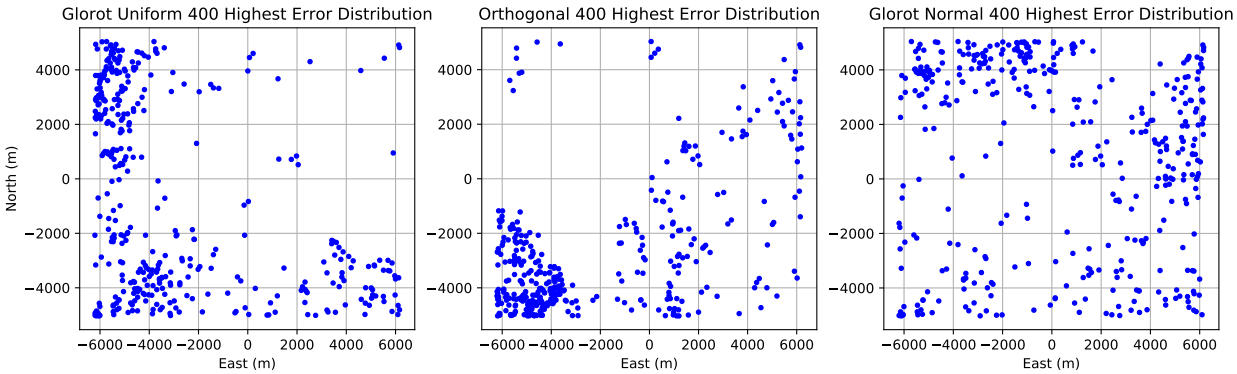

Figure 38. This scatter plot indicates the actual location of the 400 highest errors for the initializer models: Glorot uniform, orthogonal, Glorot normal.

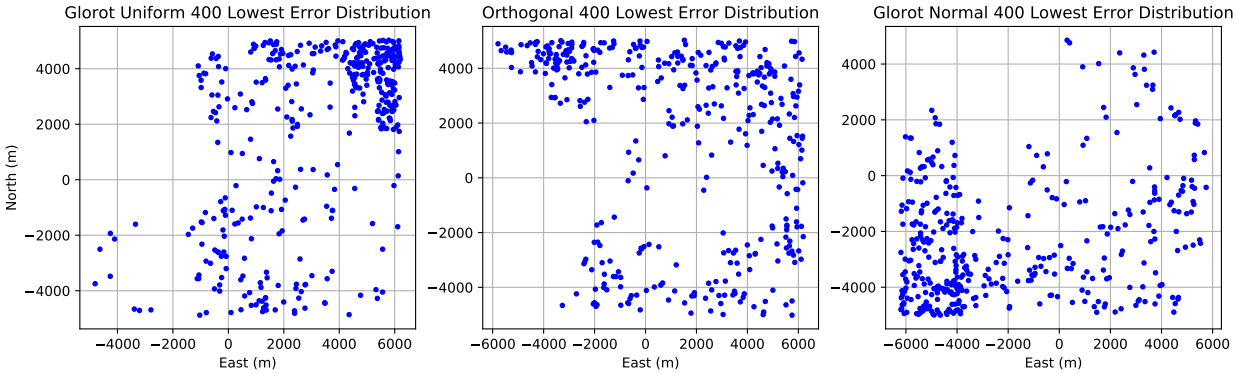

Figure 39. This scatter plot indicates the actual location of the 400 lowest errors for the initializer models: Glorot uniform, orthogonal, Glorot normal.

Weight Initializations Figure 37 displays how well Glorot Normal model performed on the test dataset compared to Glorot uniform and orthogonal. It was surprising to see how poorly orthogonal performed, given the performance in training and validation. The inability of the orthogonal network to generalize the dataset is emphasized in the highest 400 error distribution in Figure 38. The majority of the error comes from a small South-West segment of the bounding box. The best performance for the orthogonal model in Figure 39 contained no values in that South-West segment. Glorot Uniform also had highly regional error rates in both the worst errors and the best errors. The CNN model with the best usage of the area, Glorot normal, also had the best performance on the dataset.

Table 3. The default model is listed first with Adadelta optimizer, global average pooling finishing layer and Glorot uniform weight initializations. The super-model had RMS prop optimizer, flatten finishing layer, and orthogonal weight initializations.

Test Set Frobenius Norm Error after 150 Training Epochs (Meters)				
Model Type	Mean Error	Median	Max Error	Min Error
Default Model	657.470	446.707	12076.074	2.892
Super Model	299.397	220.447	12413.668	0.844

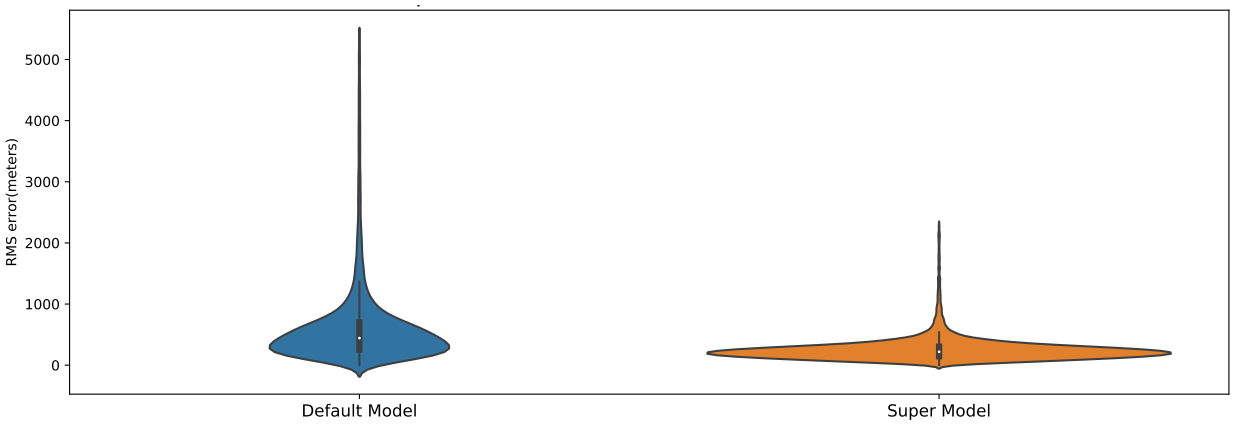

Figure 40. This violin plot displays the RMS error results of the test dataset for the weight initializer comparison models: Glorot uniform, orthogonal, and Glorot normal. Each model was trained for 50 epochs, and the worst 200 RMS errors were removed for readability.

Figure 41. Images of the 50 highest test set errors for the default model(Adadelta optimizer, global average pooling finishing layer, and Glorot uniform weight initialization) after being trained for 150 epochs.

4.2.2.2 Super Model Testing

After training the default model (Adadelta, global average pooling, and Glorot uniform) and the super-model (RMS prop, flatten, and orthogonal) for 150 epochs it was evident that the super-model had superior validation performance throughout

Figure 42. Images of the 50 highest test set errors for the super-model(RMS prop optimizer, flatten finishing layer, and orthogonal weight initialization) after being trained for 150 epochs.

training. Similarly to the models trained for 50 epochs, the worst errors were in the range of the earlier models, as seen in Table 3. The mean test set error is where the super-model truly shined. The default model had RMS error rate of 657.5 meters, where the super-model was at 299.4 meters. The super-model's error rates are nearly viable for practical application, whereas the default model is not. The errors are visualized in the violin plot in Figure 40. The default model's distribution is prone to much higher errors than the super-model.

The images of the 50 highest errors for the two networks are displayed in Figures 41 and 42. Both figures exhibit the effect of cloud cover on the dataset. The default model's worst 50, in Figure 41, suggests that the network continues to have difficulty with specific types of images, particularly in neighborhoods of dense buildings. Observing the difference between Figures 41 and 42 indicates that the default model has classified unreadable images better than images with identifiable features. The super-model has similar issues with misclassification, but at a lesser rate. Nine

of the fifty images in Figure 42 are unreadable, and classification is impossible. There are four additional images with heavy cloud cover where classification is difficult at best. These issues indicate the real obstacles that occur in visual aerial localization.

4.3 CNN Model Architecture Comparison

The CNN model comparison analyzes the training and performance of five innovative CNN architecture designs: MobileNet V2, Inception V3, ResNet 50, Xception, and DenseNet 201. These five networks were trained twice for 150 epochs, first with pretrained Imagenet weights, then with untrained Glorot uniform initializations.

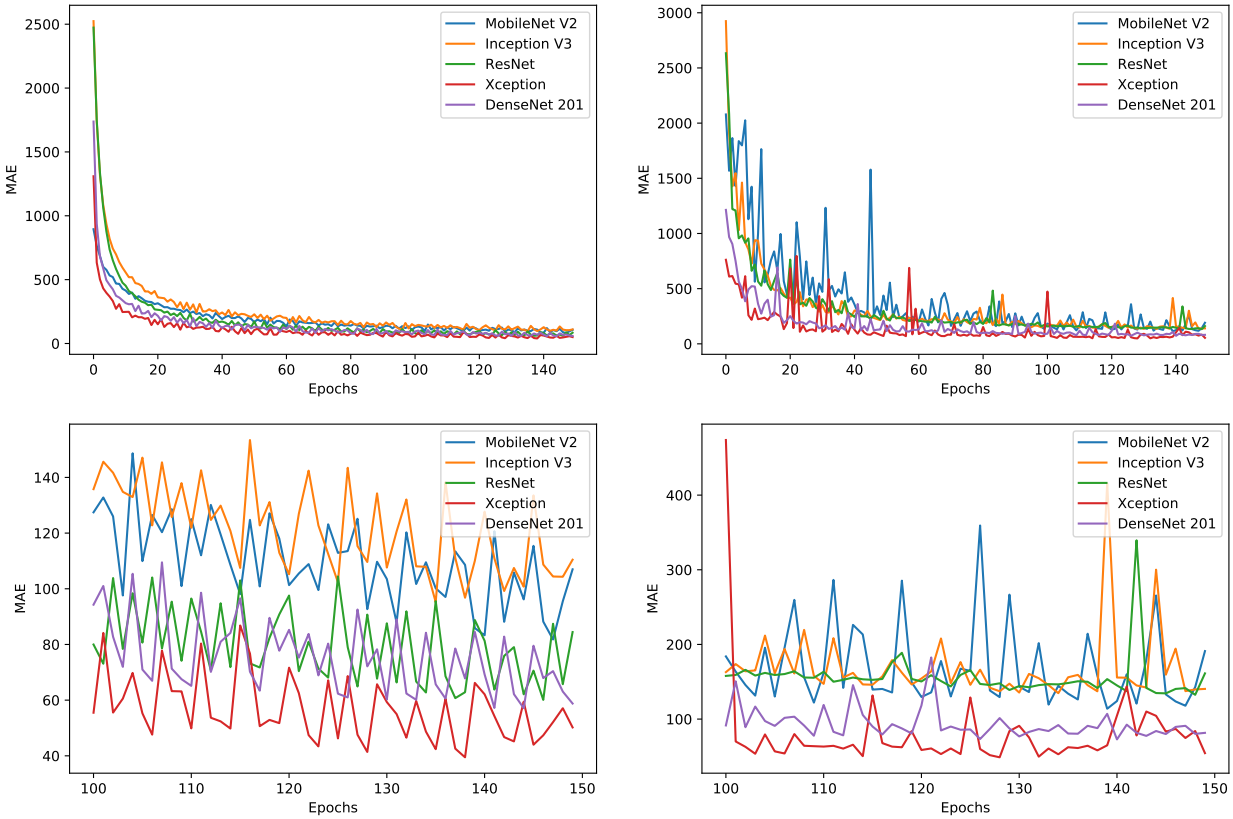

Figure 43. The training and validation error for models initialized with pretrained Imagenet weights. The model architectures are MobileNet V2, Inception V3, ResNet, Xception, and DenseNet 201. The charts on the top are training over the full 150 epochs, and the charts on the bottom are a zoom of the data epochs 100 to 150. The left charts are training errors, and the right are the validation errors for both models over the epochs.

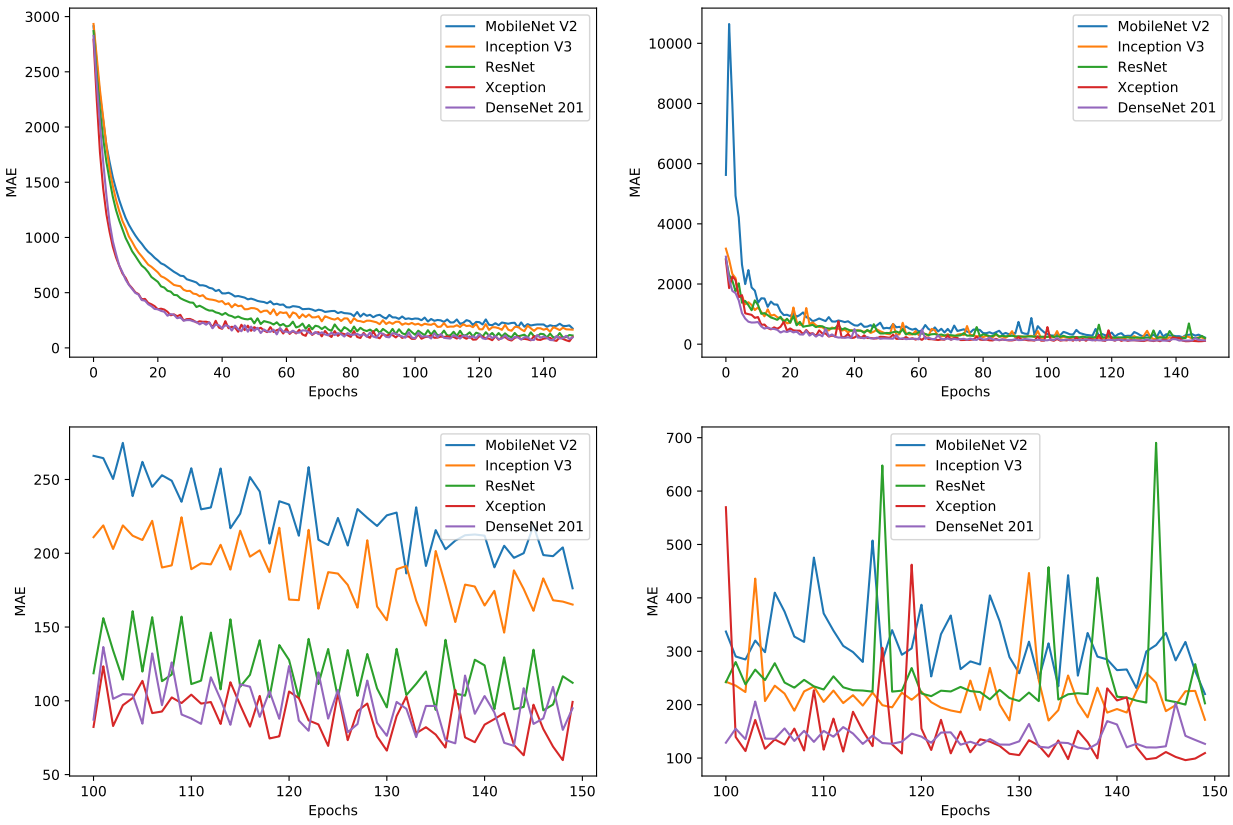

Figure 44. The training and validation error for models initialized with untrained Glorot uniform initialization. The model architectures are MobileNet V2, Inception V3, ResNet, Xception, and DenseNet 201. The charts on the top are training over the full 150 epochs, and the charts on the bottom are a zoom of the data epochs 100 to 150. The left charts are training errors, and the right are the validation errors for both models over the epochs.

4.3.1 Training

The training and validation comparison between the five models are found in Figures 43 and 44. Figure 43 provides a side-by-side chart of the models trained with Imagenet weights. The validation Mean Absolute Error (MAE) for every model with the pretrained initialization is lower than the MAE of the super-model from the hyper-parameter study after 150 epochs. Each model with pretrained Imagenet initializations also outperformed the same model with untrained Glorot uniform initializations, see Figure 44. Imagenet classification is a significantly different problem than aerial visual localization, but these pretrained weights outperform untrained

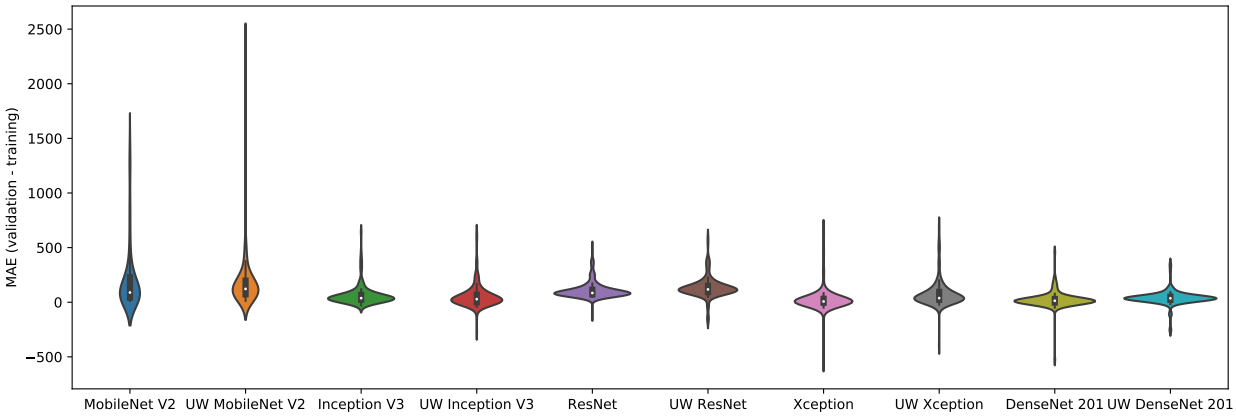

Figure 45. Validation minus training error for models side by side initialized with pretrained Imagenet weights and untrained weights (UW on the chart). The model architectures are MobileNet V2, Inception V3, ResNet, Xception, and DenseNet 201. Untrained weights MobileNet V2 contains outliers beyond the chart.

initializations, demonstrating that there is some valid transfer learning that can be extracted from the classification problem.

The side by side violin plot in Figure 45 graphically demonstrates the distribution of the validation MAE minus the training MAE. Each model's validation error distribution is only marginally better than the untrained weight initializations. This indicates that through additional training untrained initializations could have similar results.

The Xception and DenseNet 201 models had the lowest validation error of the model's trained, both with pretrained weights and untrained. The Xception model had lower validation errors than DenseNet 201 when pretrained weights were used. Although, the validation errors were nearly even when untrained initializations were used. DenseNet 201 and Xception were the most effective in generalizing the training data. MobileNet V2 consistently had higher validation errors than the others, but they were within a reasonable margin of error from the other CNNs. Inception V3 and ResNet were not far off the leaders by the end of the validations. As seen in the hyper-parameter comparison, better validation errors do not necessarily transfer into an accurate network on the test set.

4.3.2 Testing

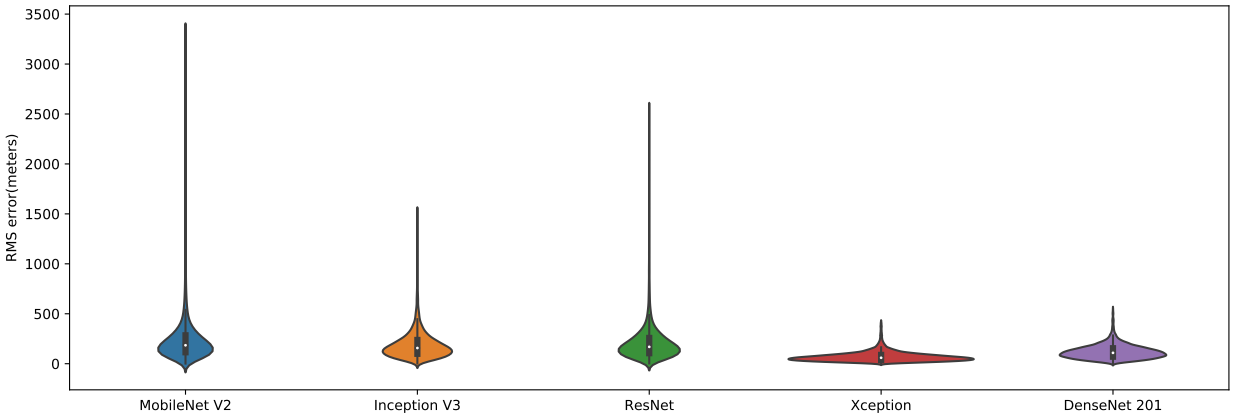

Figure 46. Prediction RMS Error results for Models: MobileNet V2, Inception V3, ResNet 50, Xception, and DenseNet 201. Each model initialized with pretrained Imagenet weights and trained 150 epochs.

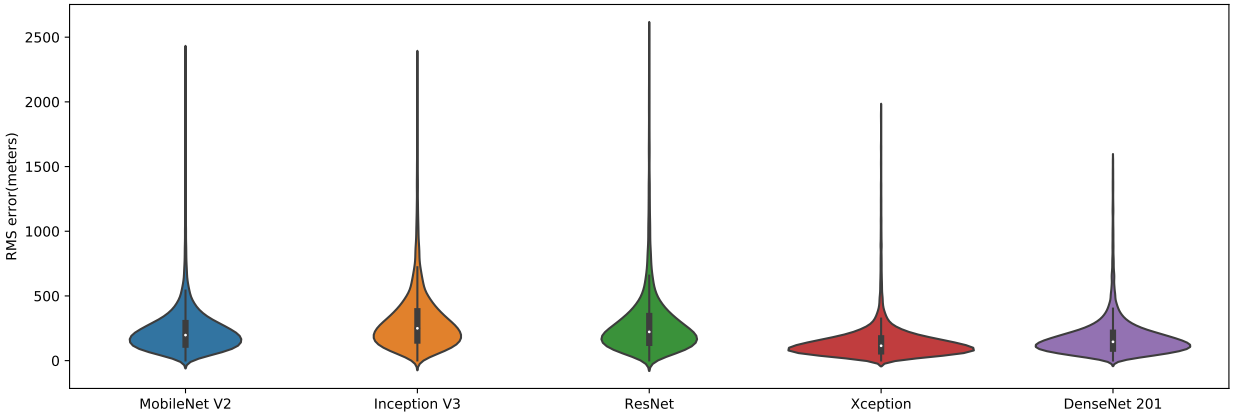

Figure 47. Prediction RMS Error results for Models: MobileNet V2, Inception V3, ResNet 50, Xception, and DenseNet 201. Each model initialized with untrained Glorot uniform weights and trained 150 epochs.

The performance on the test dataset was similar to that during training for each CNN model. Table 4 shows the RMS error statistics for each model trained. The mobileNet V2 was the worst performer, followed by ResNet 50. There were no models that performed exceptionally well without indicating during training. The performance of Xception exceeded expectations. Xception's mean RMS error was 85.5 meters after training with pretrained Imagenet weights. Each model trained with

Table 4. Each model's weights were intialized with either pretrained Imagenet weights (Imagenet), or untrained Glorot uniform weight initializations (Untrained) prior to 150 epochs of training. This displays pertinent statistics relating to each model's performance on the test dataset.

Test Set Frobenius Norm Error after 150 Training Epochs (Meters)				
Model Type (Initializer)	Mean Error	Median	Max Error	Min Error
MobileNet V2 (Imagenet)	307.307	186.034	8813.559	1.494
MobileNet V2 (Untrained)	348.788	251.728	8735.828	1.648
InceptionV3 (Imagenet)	222.459	158.414	8349.126	2.122
InceptionV3 (Untrained)	277.351	198.821	6909.972	1.572
ResNet 50 (Imagenet)	262.674	170.082	8834.585	2.985
ResNet 50 (Untrained)	330.408	224.358	8129.125	1.951
Xception (Imagenet)	85.492	59.791	10524.034	0.416
Xception (Untrained)	179.109	115.716	9515.350	1.223
DenseNet 201 (Imagenet)	129.266	109.439	6297.735	1.279
DenseNet 201 (Untrained)	203.251	146.420	7894.755	1.025

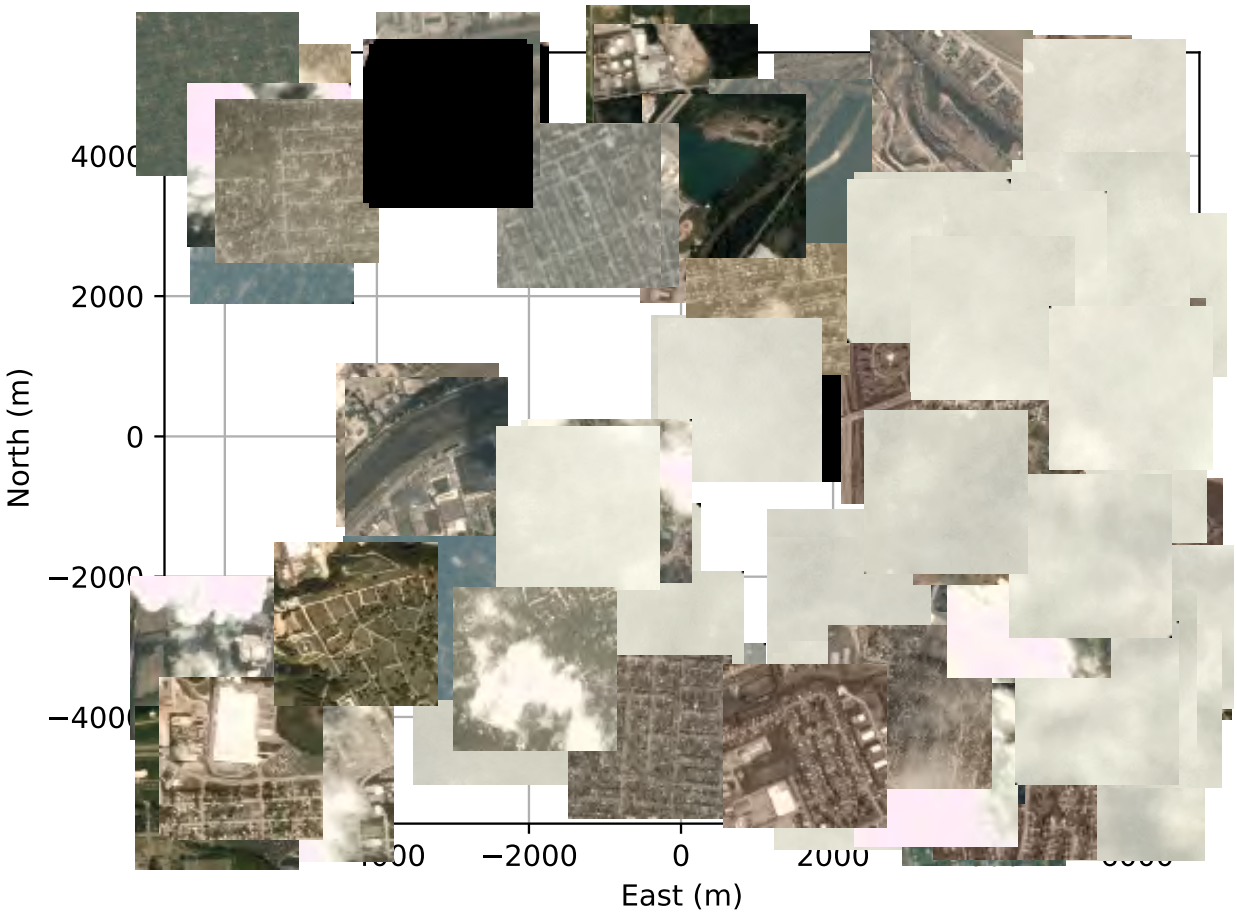

Figure 48. Xception with Imagenet initializations; This displays the actual locations of the worst 100 test set errors for the Xception model.

pretrained Imagenet weights outperformed its sister model trained with untrained weight initializations when evaluated on the test set.

The training error distribution indicated in Figures 46 and 47 sheds further light on the difference between the various networks ability to classify coordinates properly. Xception's distribution in Figure 46 provides accurate localization data reliably. The distributions for the other models, including Xception with Glorot uniform initializations in Figure 47, require additional training to achieve admissible results.

The geographic distribution of the highest error rates for the best performing model, Xception with Imagenet initialization, is in Figure 48. This shows images

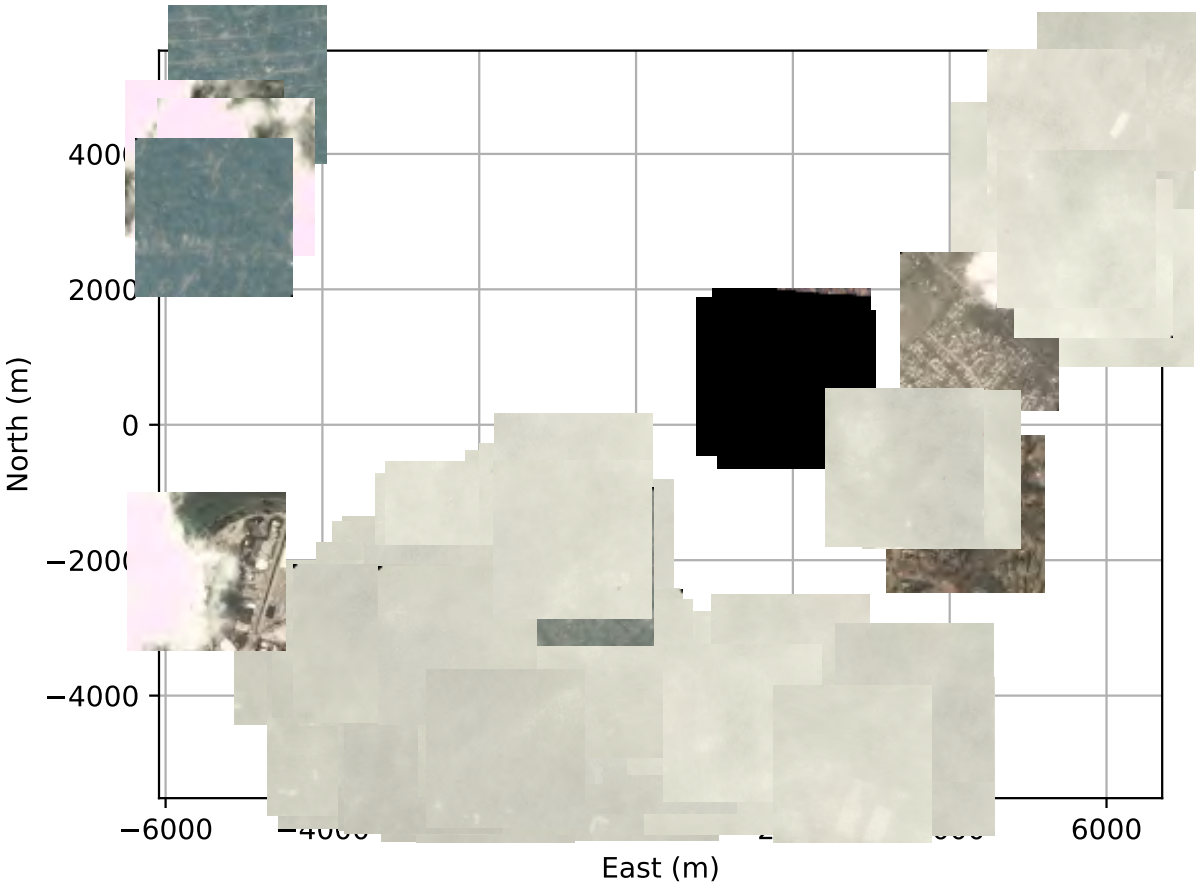

Figure 49. MobileNet V2 with Imagenet initializations; This displays the actual locations of the worst 100 test set errors for the MobileNet V2 model.

that are difficult to classify, but also portrays some images that could be classifiable, indicating that additional training could improve this network. The worst performing model with the Imagenet initialization, MobileNet V2, is shown in Figure 49. The geographic distribution of errors for MobileNet V2 displays many images that are difficult or impossible to classify. This also indicates that geographically localized errors are not necessarily bad, such as these with heavy cloud cover or unreadable images, but that we need to address them by throwing out images that cannot be localized.

4.4 Summary

The dataset contains unique traits due to the nature of processing the satellite data. These traits had an affect on the training and testing of the networks. Even with the powerful AWS instances, training time was a limiting factor, as many of the CNN models required over 20 hours to train for 150 proscribed epochs.

The Hyper-parameter Comparison discussed in Section 4.2 had diverse training and testing performance. The three hyper-parameters were optimizers, finishing layers, and weight initializers. RMS prop optimizer, flatten finishing layer, and orthogonal weight initializations performed the best during training. These three hyper-parameters were combined to create a super-model to train for 150 epochs. Testing demonstrated that Adam optimizer, flatten finishing layer, and Glorot normal performed the best. The super-model outperformed the default model, see Section 3.3, when both models were trained for 150 epochs.

Each model was trained twice the first with pretrained Imagenet weight initializations, and the second with untrained Glorot uniform weight initialization. During training, the Xception and DenseNet 201 models outperformed the others. On the test set, Xception performed the best.

V. Conclusion

Each network in this research was designed to take a single aerial image within the greater Dayton, Ohio area, and output an estimate for the geographic coordinates for that image. A study was performed on the effect hyper-parameters and model architectures can have on the training and perfomance of a CNN for aerial visual localization. An aerial image dataset was created using raw satellite images and their local coordinates. Nine hyper-parameters were studied in the areas of optimizers, finishing layers, and weight initializations. Five leading model architectures were analysed through training and evaluation on the dataset. Finally, a Minimum Mean Squared Error (MMSE) loss function was created and evaluated against the traditional Mean Squared Error (MSE) loss.

5.1 Hyper-parameter Analysis

Three hyper-parameters were varied, optimizers, finishing layers, and weight initializations. The optimizers and weight initializations had no affect on network size, but the finishing layers did. The Inception V3 model with a flatten finishing layer had 74,200,226 trainable parameters, where as the global average pooling and global max pooling models only had 23,868,578 trainable parameters. Performance on the test set did improve from the default model when using the flatten finishing layer, implying that the increased parameters may be needed to better solve the aerial localization problem. The test set RMS mean error with flatten was 295 meters lower than the global average pooling finishing layer. However, global max pooling was 116 meters higher than flatten with a significantly smaller overhead. A combination of global max pooling and global average pooling would create only a marginally larger network, but could potentially add a significant dimensionality feeding into the clas-

sification layer that may improve the performance nearly to the level that flattening provided.

Orthogonal weight initializations dramatically reduced the volatility in the validation errors during training, and training validation errors between epochs were better than the other two weight initializations: Glorot uniform and Glorot normal. Yet, this error rate did not translate into proper training, as the orthogonal initializer was out performed by both the Glorot Uniform and Glorot normal. Glorot normal initialization had the lowest test set performance of the three and the test RMS median value was 258 meters less than orthogonal, which was significant.

The optimizer, RMS prop, had reduced volatility in the validation error during training when compared to the other optimizers in the study. Regrettably, this pattern did not hold through to the test set either. RMS prop had the worst test set RMS mean error of all the models trained in the hyper-parameter study. The Adam optimizer, which had the worst training performance, did the best between the three optimizers in testing. The performance of Adadelta was in the middle of Adam and RMS prop in training and testing.

The super-model, which was designed with the best hyper-parameters on the training set (RMS prop, flatten, and orthogonal initialization) displayed how effective the additional trainable parameter could be with a fully trained network. The super-model was trained along with the default model for 150 epochs. The test set RMS error for the default model was 657 meters, where as the super-model achieved 299 meters. The additional trainable parameters provided by the flatten finishing layer had a significant effect on the performance for this dataset, once it was sufficiently trained at a level of 150 epochs. There remains the possibility that the performance observed in the hyper-parameter study could drastically change if the training could have extended further than the allowed 50 epochs. However, due to the long training

time of over 15 hours per model, it was impossible to investigate the study at a more proper level of 150 epochs.

5.2 CNN Model Architecture Comparison

Five innovative architectures were selected for this study: MobileNet V2, Inception V3, ResNet 50, Xception, and DenseNet 201. The lessons learned from the hyper-parameter analysis were implemented into this study. Each model was finished with the flatten finishing layer and trained with the Adam optimizer. Each architecture is comparable in total parameters, with MobileNet V2 as the smallest at 66,486,338 and Xception being the largest at 123,625,002. Each model was trained twice with different initializations. The first was with pretrained Imagenet weights and the second, was with an untrained Glorot uniform initialization.

Xception and DenseNet 201 (the two with the largest parameter sizes) had the lowest training validations with and without the imagenet initialization. All the other models performed similarly during training, with MobileNet V2 as the worst performer. This would lend additional credibility to a hypothesis that a large number of parameters are required for processing the data for the aerial localization task. The difference in training between the models with pretrained Imagenet weights and those with untrained initializations were significant. This provides evidence that transfer learning from the classification problem can be applied to this dataset. The error distributions were also tighter on the Imagenet models, leading to a more accurate dataset with less error.

After all the models were trained for 150 epochs they were evaluated on the test set. The Xception model with Imagenet initialization outperformed all other models with a RMS mean error of only 85 meters. The DenseNet 201, with imagenet initializations, was close behind with a RMS mean of 129 meters. The training set was designed

to produce comparable models and not accurate results, so the performance of these two models, Xception and DenseNet 201, is even more impressive. Models with pretrained weights out performed their sister models with untrained initializations, demonstrating again that there is value in the transfer learning from the categorization problem. Xception and DenseNet 201, with untrained initializations, had better test set performance than all the other models with pretrained weights. Further training with Xception and DensNet 201, with a more challenging dataset, could provide even better aerial localization.

5.3 Real World viability

This study has demonstrated models that successfully identified the location of an image within an area of interest, with an error range that could be useful given the availability of other sensor data with which to fuse. With additional training, and varying the dataset, these models could aid in presenting a solution in a Global Navigation Satellite Systems (GNSS)-denied environment. The Xception model had a Euclidean norm accuracy of 85.492 meters. With each image being 700 meters from edge to edge the average image center location is well within the image.

5.4 Future Work

Some specific was to build on this work would include: comprehensive network training, utilizing flight test data, and integrating additional sensors. Advanced network training can develop networks that understand the area more completely. Flight test data would train the network for real world implementation, and integrating into additional sensors will develop a robust and reliable aerial visual localization solution.

Training a network with on-demand dataset generations can aid the CNN in better generalizing the information provided. Enhancing the sample images with varied

hues and saturations would further diversify the dataset, by simulating different lighting conditions and operational environments. Combining the CNN with a Recurrent Neural Network (RNN) and running simulated flight paths could further develop consistently better accuracies. Finally, creating a dataset that integrates information from additional flight instrumentation into the predictive network could create impressive accuracies.

Implementing these CNNs into a flight based system is the ultimate goal, and being that, testing and training the networks with actual flight imagery is the next step. Flight test imagery can be used through every step of the training and evaluating the networks. Varying altitudes and challenging the model's ability to adapt, would present opportunities to add additional design features or limitations to each model.

Finally, CNNs are only a small piece in a complete solution to GNSS-denied aerial localization. Flight sensors and instrumentation must be integrated with the model results to generate solutions that can be generated and validated. This study has demonstrated justification for further research to the viability of CNNs as a contributing factor of a larger solution for aerial localization .

Appendix A. MMSE Loss

1.1 Abstract

A new MMSE loss function takes the minimum error of four estimates from the network. This loss is designed to develop areas with more features and better regionalization, while being combined with additional sensors such as an Inertial Measurement Unit (IMU) for comprehensive avigation solutions. This loss function will require additional research to reach state of the art capability.

1.2 Methodology

All of the networks previously studied have one input and one set of output coordinates. This forces the network to select one location based on the image. In real world flight there are many estimation techniques that can eliminate outliers and incorrect location coordinates. The ability to have multiple location choices in different areas could be useful to an integrating algorithm that selects the most plausible location based off of IMU estimations against a limited number of location choices.

This section will overview the design of a custom MMSE loss function that could be used to develop a model with four location choices. If developed properly, the selections could be integrated into a navigation algorithm to provide a more accurate location determination than a network trained using a single MSE loss function. The loss will follow the MSE structure, but instead only return the minimum error per instance while the other errors will be reduced to zero.

The model will be trained with similar techniques as those used in section 3.4. MMSE will be the loss, Minimum Mean Absolute Error (MMAE) will be developed as a metric along with traditional MAE to compare the total error versus the minimum error. The training and performance of this network will be compared to the

previous comparisons, and the viability of implementation in a navigation system will be analyzed.

1.2.1 Minimum Mean Squared Error Loss Function

Navigation using IMUs is well defined[57]. The practice of updating ones position based on the airspeed, the direction, and the altitude is fairly reliable for a short time. A system advanced enough to process an image successfully has ability to easily process IMU information. MMSE provides multiple location estimates and updates, in this case it is four. Why use four location estimates? The redundancy of data can easily remove incorrect guesses over time as long as the guesses are diverse and not clustered in one area. Avigation is completed using more than one instrument, it is a combination of multiple instruments over time to determine one's position. The camera comparison on a CNN will provide one aspect of navigation and the IMU can provide another. A filter can be utilized to fuse the outputs together.

The problem with outputting four location estimates is that a loss function with multiple outputs will hold the network accountable for each of the losses together, and each loss will not be diverse in its location, causing each location output to mirror itself. To solve this problem, a loss function must be designed to only care about the minimum loss and ignore the rest of the losses.

MSE was selected because of the property of being a linear differentiable loss function, in which gradient descent could be used. Also, because MSE is squared, it punishes rapidly as the error gets further from the correct location. The first step in developing the MMSE loss function is to take the square error of each output location. The function then reshapes the output and determines the minimum loss for each output value. After the minimum loss is calculated, a mask is created so only the minimum loss remains and the higher losses resort to zero. Finally, the final

error's mean is provided to the optimization function.

1.2.2 Specific Design

The model selected for training with the MMSE loss function will be the Inception V3. Inception V3 was utilized in the hyper-parameter comparison as the default, and performance in model comparison study adequately justified its continued use as a default model. The pretrained Imagenet weights will be used because their performance exceeded the performance of untrained weights. The other conditions are the same as the model comparison as discussed in section 3.4.

The model will be trained using MMSE and MMAE will be used as the metric. MMAE uses the same design structure as MMSE with the exception of taking the absolute error instead of the square error. Traditional MAE will also be reported to compare over training.

1.2.3 Measuring Performance

The objective of this new loss function is to determine one accurate location and three locations that are obvious outliers. The success of this function can be determined if MMAE is low and MAE is high. A high MAE with a low MMAE indicates that one location is close, and the remaining three are not in the area. This will also aid algorithms and filters to easily select the correct location from the model output based off IMU measurements from the last known location.

The model will be compared with the same model trained on a traditional MSE loss. The performance errors will be analyzed to determine if the model is maintaining diversified locations for each value. MMAE vs MAE will be compared. The highest image errors will be visually analyzed. The geographic position of highest errors will be reviewed and compared to previous studies.

1.3 Results

The MMSE loss function had a very rapid training rate. The training and validation exceeded the model trained with the traditional MSE loss function. Unfortunately the training did not translate into performance. The results of investigating this new loss function are detailed in the next few sections

1.3.1 Training

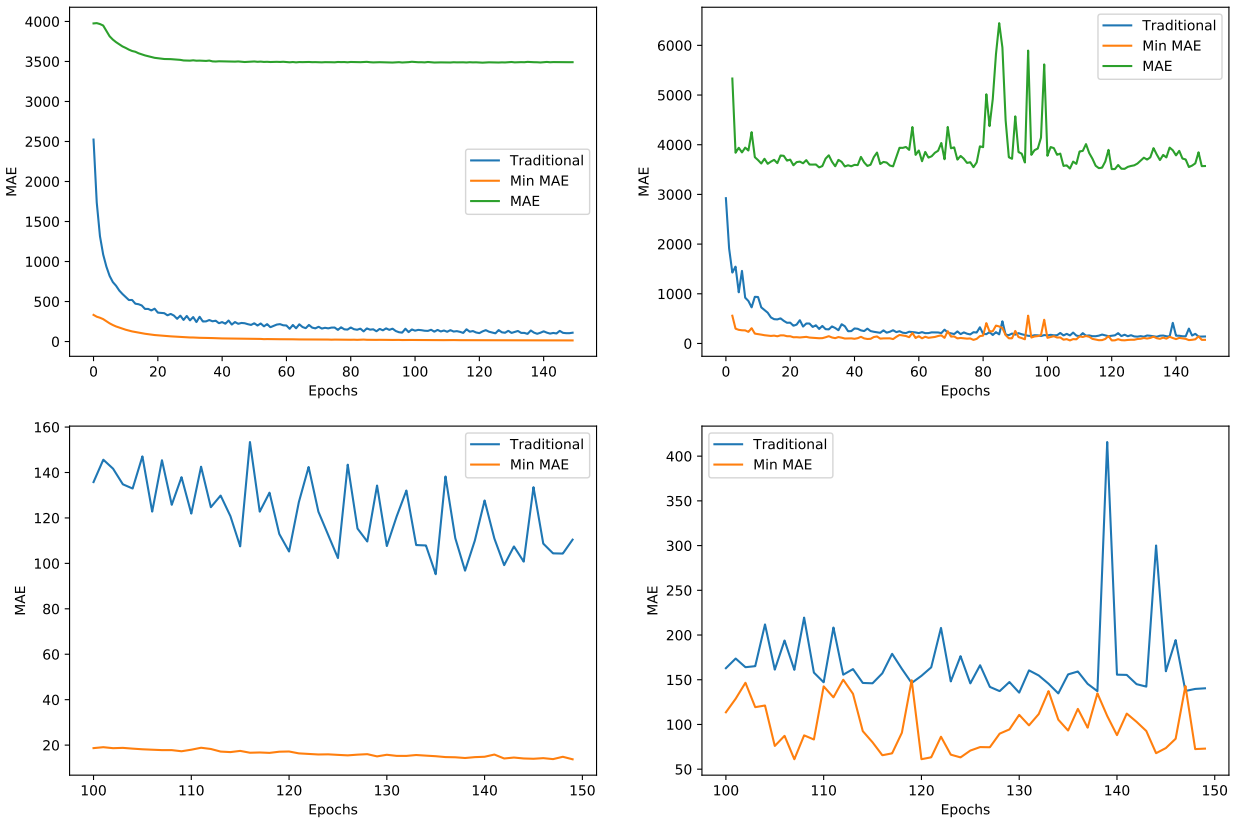

Figure 50. This chart compares the traditional training error model and the MMAE custom loss model with the actual MAE for the custom loss model.

The custom loss was tested on the Inception model using the same parameters as the Imagenet initialization Inception V3 in the Model Comparison section 4.3. The interesting phenomenon observed during training the MMSE loss function, is the difference between the MMAE and the MAE. The training chart in Figure 50

demonstrates the MAE remains high with minimal improvement while the MMAE continues to improve throughout training. This means that while there are four guesses only one of the guesses is correct and the other three are not even in the ballpark. This is a beneficial phenomenon when combining this model with the output of an IMU to aid in predicting locations. During training the best guess in validation is consistently better than the traditional model.

1.3.2 Performance

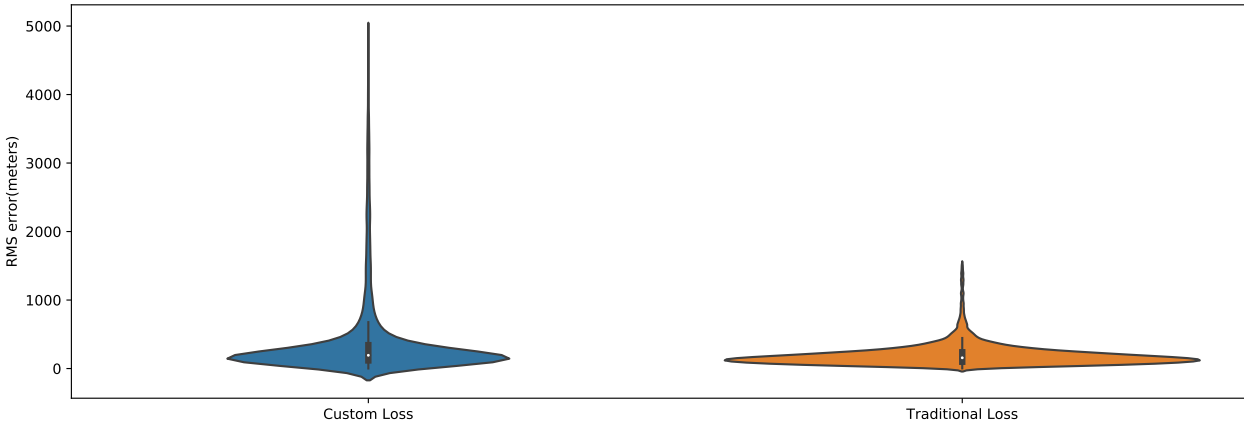

Figure 51. The custom Loss displays the RMS Test error for the model trained with MMSE loss function. The traditional loss utilized MSE loss.

The performance of the custom loss function was not on par with the training curves. Figure 51 displays The MMSE loss model's poor performance. The failure to surpass the MSE trained model could be attributed to over training with the training set. Utilizing a dataset generator as opposed to a fixed dataset could increase

Table 5. Both model's weights were intialized with pretrained Imagenet weights prior to 150 epochs of training. This displays pertinent test set statistics relating to the Inception V3 model trained with MMSE loss vs the one trained with MSE loss.

Test Set RMS Error after 150 Training Epochs (Meters)				
Model Type	Mean Error	Median	Max Error	Min Error
MMSE Loss	471.723	195.077	26801.855	1.151
MSE Loss	222.459	158.414	8349.126	2.122

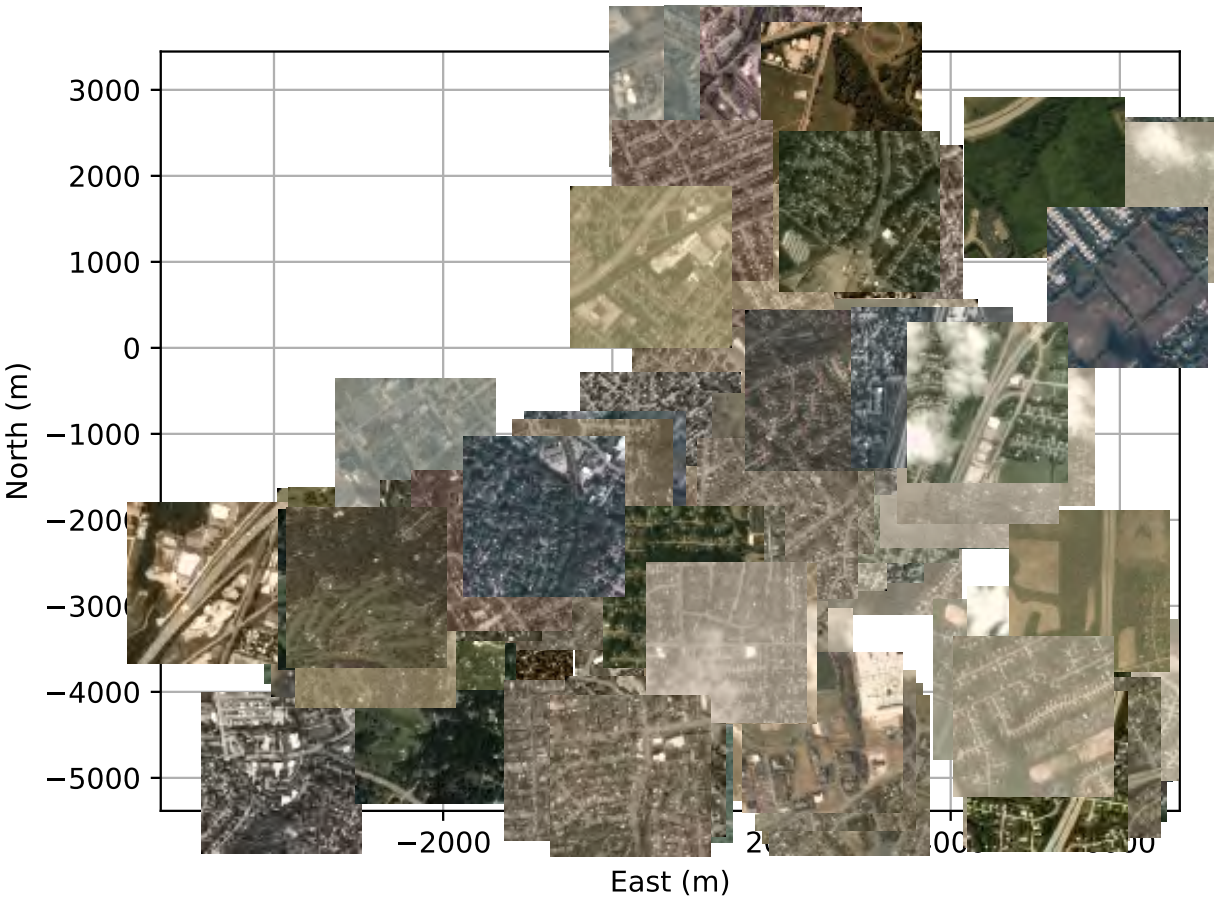

Figure 52. The Inception V3 model trained with custom MMSE Loss highest 100 geographic error disstribution.

the performance of the MMSE loss function. Table 5 provides information that the MMSE failed to lower maximum errors with its multiple guesses. The max error of 26,801 it is higher than all the other models trained for 150 epochs in this study. Figure 52 displays the failure of the MMSE model to select locations for images with clearly defined features, which is further evidence of over-training. Additional training modifications to diversify the dataset during training would aid this network in providing more reliable coordinates on the test set and reduce over-training.

1.4 Conclusion

MMSE loss function took the minimum error of four guesses. This loss is designed to develop areas with more features and better regionalization. During training the loss quickly progressed to local minimum and produces better validation errors consistently over the traditional MSE loss. An indication of the model selecting one 'in the money' guess with the rest out of range, is the total MAE of the function which remained high throughout the training as the MMSE improved.

Sadly, the training performance did not translate into test performance. The model trained with MMSE underperformed on the test dataset and had much higher errors. This phenomenon indicated that the MMSE loss model was overtrained and memorized the training data as opposed to generalizing the data. In order to test this hypothesis, additional training and dataset diversification needs to take place. Training helpers, such as dropout, could aid to force the network to better generalize the data.

Bibliography

1. Arthur J. Hughes. *History of Air Navigation*. Unwin Brothers Limited Woking, London, 1st edition, 1946.

2. Alan Grant, Paul Williams, Nick Ward, and Sally Basker. GPS jamming and the impact on maritime navigation. *Journal of Navigation*, 62(2):173–187, 2009.

3. Farhad Samadzadegan and Ghasem Abdi. Autonomous Navigation of Unmanned Aerial Vehicles Based on Multi-Sensor Data Fusion. In *20th Iranian Conference on Electrical Engineering*, pages 868–873, 2012.

4. T Machin, John F Raquet, D Jacques, and D Venable. Real-time implementation of vision-aided navigation for small fixed-wing unmanned aerial systems. *29th International Technical Meeting of the Satellite Division of the Institute of Navigation, ION GNSS 2016*, 2:1305–1311, 2016.

5. Donald T. Venable and John F. Raquet. Large scale image aided navigation. *IEEE Transactions on Aerospace and Electronic Systems*, 52(6):2849–2860, 2016.

6. Girish Chowdhary, Eric N Johnson, Daniel Magree, Allen Wu, and Andy Shein. GPS-denied Indoor and Outdoor Monocular Vision Aided Navigation and Control of Unmanned Aircraft. 30(3):415–438, 2013.

7. Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. *Proceedings - IEEE International Conference on Robotics and Automation*, (May):15–22, 2014.

8. Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation. In *Robotics: Science and Systems XI*, 2015.

9. Stephen Williams, Vadim Indelman, Michael Kaess, Richard Roberts, John J. Leanoard, and Frank Dellaert. Concurrent filtering and smoothing: A parallel architecture for real-time navigation and full smoothing. *International Journal of Robotics Research*, 33(12):1544–1568, 2014.

10. Antoni Rosinol Vidal, Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. Hybrid, Frame and Event based Visual Inertial Odometry for Robust, Autonomous Navigation of Quadrotors. pages 1–10, 2017.

11. Vasko Sazdovski, Andrej Kitanov, and Ivan Petrovic. Implicit observation model for vision aided inertial navigation of aerial vehicles using single camera vector observations. *Aerospace Science and Technology*, 40:33–46, 2015.

12. He Bai and Clark N Taylor. Control-enabled Observability in Visual-Inertial Odometry. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, Miami, FL, 2017.

13. Huangying Zhan, Ravi Garg, Chamara Saroj Weerasekera, Kejie Li, Harsh Agarwal, and Ian Reid. Unsupervised Learning of Monocular Depth Estimation and Visual Odometry with Deep Feature Reconstruction. *Computer Vision and Pattern Recognition (CVPR), 2018*, 2018.

14. Ruihao Li, Sen Wang, Zhiqiang Long, and Dongbing Gu. UnDeepVO: Monocular Visual Odometry through Unsupervised Deep Learning. *arXiv preprint*, 2017.

15. Stefan Jorgensen, Niveta Iyer, Chelsea Sidrane, and Csidrane@stanford Edu. Teach me to Tango: Fidelity Estimation of Visual Odometry Systems for Robust Navigation. Technical report, 2017.

16. Hongkai Wen, Sen Wang, Ronald Clark, and Niki Trigoni. {DeepVO}: Towards End to End Visual Odometry with Deep Recurrent Convolutional Neural Net-

works. *Proceedings of the {IEEE} International Conference on Robotics and Automation ({ICRA})*, pages 2043–2050, 2017.

17. Ganesh Iyer, J. Krishna Murthy, Gunshi Gupta, K. Madhava Krishna, and Liam Paull. Geometric Consistency for Self-Supervised End-to-End Visual Odometry. *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

18. Tsung-Yi Lin, Yin Cui, Serge Belongie, James Hays, and Cornell Tech. Learning Deep Representations for Ground- to- Aerial Geolocalization. page 270.

19. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks Alex. *Advances In Neural Information Processing Systems*, pages 1097–1105, 2012.

20. Andrew M Saxe, James L Mcclelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. Technical report, 2014.

21. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:1026–1034, 2015.

22. Diederik P Kingma and Jimmy Lei Ba. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. *CoRR*, abs/1412.6, 2014.

23. Sebastian Ruder. An overview of gradient descent optimization algorithms. sep 2016.

24. Alec Radford, Luke Metz, and Soumith Chintala. UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS. *CoRR*, abs/1511.0, 2015.

25. Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to End Learning for Self-Driving Cars. Technical report, 2016.

26. Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid Scene Parsing Network. Technical report.

27. Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. *CoRR*, abs/1312.4, 2013.

28. Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. Technical report.

29. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June:1–9, 2015.

30. Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. Technical report.

31. Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:2261–2269, 2017.

32. Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Pmlr*, 9:249–256, 2010.

33. Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient BackProp. pages 9–50. 1998.

34. John Duchi JDUCHI and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization * Elad Hazan. Technical report, 2011.

35. Matthew D Zeiler. ADADELTA: AN ADAPTIVE LEARNING RATE METHOD. *CoRR*, abs/1212.5, 2012.

36. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Chapel Hill, and Ann Arbor. Going Deeper with Convolutions. pages 1–9, 2014.

37. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. Technical report.

38. Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. feb 2015.

39. Rob Zimmerman, Deanna Doan, Lawrence Leung, James Mason, Nate Parsons, and Kam Shahid. Commissioning the World's Largest Satellite Constellation. *AIAA/USU Conference on Small Satellites*, 31:1–10, 2017.

40. Logan C. Ramsey. *The Navigation of Aircraft*. The Ronald Press Company, New York, New York, USA, 1929.

41. M. Li and A. I. Mourikis. High-precision, consistent EKF-based visual-inertial odometry. *The International Journal of Robotics Research*, 32(6):690–711, 2013.

42. Jakob Engel, Thomas Sch, and Daniel Cremers. LSD-SLAM: Large-scale Direct Monocular SLAM. In *European Conference on Computer Vision*, pages 834–849, 2014.

43. Aveek K Das, Student Member, Rafael Fierro, Vijay Kumar, Senior Member, James P Ostrowski, John Spletzer, and Camillo J Taylor. A Vision-Based Formation Control Framework. *Ieee Transactions on Robotics and Automation*, 18(5):813–825, 2002.

44. Natalie Frietsch, Oliver Meister, Justus Seibold, and Gert F Trommer. Vision Based Hovering and Landing System for a VTOL-MAV with Geo-Localization Capabilities. *Control*, (August), 2008.

45. N Frietsch, O Meister, C Schlaile, J Seibold, and G F Trommer. VISION BASED LANDING SYSTEM FOR A VTOL-MAV.

46. Mikel M Miller and Andrey Soloviev. Navigation in GPS Denied Environments : Feature-Aided Inertial Systems Navigation in GPS Denied Environments :. (January), 2011.

47. Jakob Engel, Jurgen Sturm, and Daniel Cremers. Semi-dense visual odometry for a monocular camera. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1449–1456, Sydney, Australia, dec 2013.

48. Noha Radwan, Abhinav Valada, and Wolfram Burgard. VLocNet++: Deep Multitask Learning for Semantic Visual Localization and Odometry. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):4407–4414, 2018.

49. Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. Geometry-Aware Learning of Maps for Camera Localization. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017.

50. Marcel Geppert, Peidong Liu, Zhaopeng Cui, Marc Pollefeys, and Torsten Sattler. Efficient 2D-3D Matching for Multi-Camera Visual Localization. 2018.

51. John A Volpe. Vulnerability Assessment of the Transportation Infrastructure Relying on the Global Positioning System. *Final Report*, page 99, 2001.

52. FAA. Navigation Programs VOR MON, 2018.

53. John M Dow, · R E Neilan, and · C Rizos. The International GNSS Service in a changing landscape of Global Navigation Satellite Systems. 83:191–198, 2009.

54. NGA. NATIONAL GEOSPATIAL-INTELLIGENCE AGENCY (NGA) STANDARDIZATION DOCUMENT DEPARTMENT OF DEFENSE WORLD GEODETIC SYSTEM 1984 Its Definition and Relationships with Local Geodetic Systems. Technical report.

55. Yansen Wang, Giap Huynh, and Chatt Williamson. Integration of Google Maps/Earth with microscale meteorology models and data visualization. *Computers and Geosciences*, 61(August):23–31, 2013.

56. Guowei Cai, Ben M. Chen, and Tong Heng Lee. *Unmanned Rotorcraft Systems*. Springer-Verlag London, 1 edition, 2011.

57. David Titterton and John Weston. *Strapdown Inertial Navigation Technology*. Revised, 2 edition, 2004.

58. Toshio Fukushima. Transformation from Cartesian to Geodetic Coordinates Accelerated by Halley's Method. *Journal of Geodesy*, (79):689–693, 2006.

59. James Ackeret, Fred Esch, Chris Gard, Frederick Gloeckler, Daniel Oimen, Juan Perez, Justin Simpson, Daniel Specht, David Stoner, Joe Watts, Tom Witte, and Harry Rossander. Handbook for Transformation of Datums, Projections, Grids,

and Common Coordinate Systems Topographic Engineering Center. Technical report, 2004.

60. I Goodfellow, Y Bengio, and A Courville. Deep learning. *Healthcare Informatics Research*, 22(4):351–354, 2016.

61. Warren S. McCulloch and Walter H. Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The bulletin of mathematical biophysics*, 5.4:115–133, 1943.

62. Zahangir Alom, Tarek M Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Essen, Abdul A S Awwal, and Vijayan K Asari. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. *CoRR*, abs/1803.0, 2018.

63. Marvin L Minsky and Seymour A Papert. *Perceptrons*. Massachusetts, 2nd edition, 1969.

64. Ben Krose and Patrick van der Smagt. *An Introduction to Neual Networks*. Amsterdam, 8th edition, 1996.

65. D 0 Hebb. *The Organization of Behavior A NEUROPSYCHOLOGICAL THEORY*. John Wiley & Sons, Inc., New York, New York, USA, 1st edition, 1949.

66. KUNIHIKO FUKUSHIMA. Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition. *Neural Networks*, 1(2):119–130, 1988.

67. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. 2015.

68. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Technical report, 2014.

69. Dmytro Mishkin and Jiri Matas. ALL YOU NEED IS A GOOD INIT. Technical report.

70. Jia-Ren Chang and Yong-Sheng Chen. Batch-normalized Maxout Network in Network. Technical report.

71. Keras. Initializers - Keras Documentation.

72. Sebastian Ruder. An overview of gradient descent optimization algorithms *. Technical report.

73. Christian Daniel, Jonathan Taylor, and Sebastian Nowozin. Learning Step Size Controllers for Robust Neural Network Training. *Proceedings of the 30th Conference on Artificial Intelligence (AAAI 2016)*, pages 1519–1525, 2016.

74. Mahesh Chandra Mukkamala and Matthias Hein. Variants of RMSProp and Adagrad with Logarithmic Regret Bounds. Technical report, 2017.

75. Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning Deep Features for Discriminative Localization. Technical report.

76. Francois Chollet. *Deep Learning with Python*. Manning, Shelter Island, NY, 1 edition, 2017.

77. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017.

78. Xiangyu Zhang, Xinyu Zhou, and Mengxiao Lin. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. Technical report.

79. DJI. DJI Phantom 4 User Manual. Technical report, DJI, 2017.

80. Travis E Oliphant. Python for Scientific Computing. *IEEE Computing in Schience & Engineering*, pages 10–20, 2007.

81. Jean Francois Puget. The Most Popular Language For Machine Learning Is ... (IT Best Kept Secret Is Optimization). 2016.

82. Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, and Google Brain. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, pages 265–283, Savannah, GA, 2016.

83. Francois Chollet. *Deep Learning with Python*. Manning, Shelter Island, NY, 1st edition, 2017.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
21–03–2019	Master's Thesis	Sept 2017 — Mar 2019

4. TITLE AND SUBTITLE

CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE STUDY FOR AERIAL VISUAL LOCALIZATION

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Jedediah Mark Berhold, Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-19-M-010

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Jeffery Herbert
937-713-4418
AFRL SENSORS DIRECTORATE
2241 Avionics Cir
WPAFB OH 45433-7765

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RYW

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research studies best practices in convolutional neural network (CNN) architecture design for a novel solution to aerial visual localization. A dataset was developed of satellite images modeling aircraft photographs with corresponding locations in an area of interest. CNN performance, based on varying hyper-parameters such as: optimizers, finishing layers, and weight initializers, was analyzed. Industry leading CNN architectures were trained and tested on the dataset and compared. Finally, a loss function was designed to facilitate returning multiple location estimations for processing with an inertial measurement unit to develop a comprehensive avigation solution.

15. SUBJECT TERMS

LaTeX,Thesis

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Robert C. Leishman, AFIT/ENG
U	U	U	UU	118	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4555; amy.magnus@afit.edu